

# 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 programarem una Taula de Hash oberta, que posteriorment fareu servir a l'exercici 4. Finalment, a l'exercici 5 comparareu les estructures i analitzareu el cost computacional de cadascuna d'elles.

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

pel·lId	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	WALLE	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();
```

# Estructura de Dades: Pràctica 3

```
/* 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 Value& v);
int depth() const;
int height() const;
bool operator==(const NODETree<Key,Value>& node) const;

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

## Pas 1 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	Destruïx els nodes fills. <i>Atenció a la gestió de memòria dinàmica</i>
getKey	Retorna la clau del node (atribut <code>Key</code> )
getValues	Retorna el vector de valors (atribut <code>Values</code> )
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. <b>Aquesta funció s'ha d'implementar de forma RECURSIVA</b>

# Estructura de Dades: Pràctica 3

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

## Pas 2 Implementar l'arbre binari de cerca

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

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

# Estructura de Dades: Pràctica 3

<b>mirrorTree</b>	Modifica l'arbre per tenir el seu arbre mirall. <b>Aquesta funció s'ha d'implementar de forma RECURSIVA</b>
<b>search</b> (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. <b>Aquesta funció s'ha d'implementar de forma ITERATIVA.</b>

## 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 = [";
```

# Estructura de Dades: Pràctica 3

```
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]

## 2. Gestor de pel·lícules (Mubiesflix) amb un arbre binari de cerca

### Pas 1 Especificació del Gestor de pel·lícules (Mubiesflix)

Implementeu la classe **MubiesflixBST** que hereti de la classe **BSTtree<int, Peli>**, on **int** sigui l'identificador d'un director. És important, a més, que l'herència sigui amb visibilitat **protected**. Això evitarà que exposeu els mètodes privats de la classe mare – és a dir, del TAD **BSTtree** – des d'aquesta nova classe **MubiesflixBST**.

La definició de l'especificació de la classe és doncs:

```
class MubiesflixBST : protected BSTtree<int, Peli> {  
public:  
    enum AdditionStrategy { AFTER_LARGEST_ID, SMALLEST_NOTTAKEN_ID };  
    MubiesflixBST(AdditionStrategy addition_strategy);  
    MubiesflixBST (AdditionStrategy addition_strategy, string file_path);  
    MubiesflixBST (const MubiesflixBST & orig);  
    virtual ~MubiesflixBST ();
```

# Estructura de Dades: Pràctica 3

```
void loadFromFile(string file_path);
void showAllPelis() const;
void showPelisByDirector(int director_id) const;
float getAverageValoracioOfDirector(int director_id) const;
void findLargestDirectorId() const;
void findSmallestNotTakenDirectorId() const;
void addPeli();
private:
    AdditionStrategy addition_strategy;
    /* Metodes auxiliars, definiu-los aquí sota */
};
```

Les operacions que el **MubiesflixBST** ha d'implementar són les següents:

Operació	Definició
<b>Constructor i constructor còpia</b>	Constructor amb un sol paràmetre (l'estratègia d'identificació de noves pel·lícules afegides), constructor amb dos paràmetre que rebi l'estratègia i la ruta del fitxer de pel·lícules per carregar a l'arbre i constructor còpia.
<b>Destructor</b>	Destructor de la classe
<b>loadFromFile</b>	Carrega les dades d'un fitxer a l'arbre. El paràmetre del mètode és la ruta on es troba el fitxer.
<b>showPelisByDirector</b>	<p>Imprimeix la llista de pel·lícules d'un director especificat pel seu <code>director_id</code>. Exemple:</p> <pre>Pel·lícula 1: peli_id1 titol1 durada1 valoracio1 Pel·lícula 2: peli_id2 titol2 durada2 valoracio2</pre> <p>Llença una excepció <code>runtime_error</code> si no existeix el director.</p>
<b>getAverageValoracioPelisByDirector</b>	Retorna la valoració mitjana de les pel·lícules d'un director especificat pel seu <code>director_id</code> .
<b>showAllPelis</b>	<p>Imprimeix les pel·lícules de tots els directors, però fent-ho de k en k directors. Després d'haver-ne imprès k, demana a l'usuari si vol veure les dels següents k directors. Exemple per k=2:</p> <pre>Director: director_id1 === Pel·lícula 1: peli_id1 titol1 durada1 valoracio1 Pel·lícula 2: peli_id2 titol2 durada2 valoracio2  Director: director_id2 === Pel·lícula 1: peli_id3 titol3 durada3 voloracio3  Vols veure les següents 2 directors? (s/n): n</pre>
<b>findLargestDirectorId</b>	Retorna el <code>director_id</code> més gran. Els identificadors de director/a són una numèric auto incremental. Si volguéssim

# Estructura de Dades: Pràctica 3

	afegir una nova pel·lícula d'un director pel qual no tenim identificador, podríem crear-li un identificador automàticament agafant l'identificador de director més gran i sumant-li 1.
<code>findSmallestNonTakenDirectorId</code>	Retorna el <code>director_id</code> petit possible que no estigui assignat a un director de <b>MubiesflixBST</b> (cal tenir en compte que la seqüència d'identificadors de directors pot contenir discontinuïtats). Posem per cas que existeixen directors amb identificadors (ordenats en ordre creixent): 0, 1, 2, 4 i 7. Llavors, aquest mètode retornaria 3.
<code>addPeli</code>	Afegeix una pel·lícula a l'arbre de manera interactiva. Se li demana a l'usuari especificar un <code>director_id</code> manualment o demanar a <b>MubiesflixBST</b> de generar-ne un automàticament segons l'estratègia d'identificació especificada pel constructor: assignar el següent identificador després del més gran o el primer més petit disponible.

Tingueu en compte que **MubiesflixBST** hereta del TAD **BSTtree** i que, per tant, teniu accés a les operacions del TAD amb visibilitat `public` i `protected`, així com també al `root` del TAD, la qual cosa us permetrà implementar qualsevol operació i re-utilitzar el màxim de codi.

## **Pas 3 Programació d'un main amb menú interactiu**

El programa main implementarà un menú d'opcions, on:

1. Oferirà a l'usuari especificar la ruta del fitxer que es vulgui carregar a Mubiesflix. Us proporcionarem, mitjançant el Campus Virtual, diversos fitxers `pelis-*.txt` que haureu d'afegir al directori arrel del vostre projecte (allà on tingueu el `main.cpp`).
2. Mostrarà les pel·lícules d'un director concret.
3. Mostrarà la valoració mitjana de les pel·lícules d'un director concret.
4. Mostrarà tots els directors ordenats pel seu identificador i les seves pel·lícules.
5. Mostrarà l'identificador més gran assignat als directors de Mubiesflix.
6. Mostrarà l'identificador més petit no assignat a cap dels directors de Mubiesflix.
7. Permetrà afegir una nova pel·lícula a un director. Per afegir-la a un director ja existent a Mubiesflix, s'haurà d'especificar l'identificador del director manualment. En cas de ser un nou director, es pot especificar manualment un nou identificador o trobar-ne un de manera automàtica seguint l'estratègia d'assignació pre-establerta.
8. Establir una estratègia alternativa d'assignació automàtica d'identificadors de nous directors.
9. Sortir

## **Cas de prova**

Us suggerim que carregueu el fitxer `pelis-cas_de_prova.txt`. I executeu les diverses operacions que se us demanen per comprovar que funcionen correctament abans d'implementar-les en el menú d'opcions. La majoria de comprovacions són trivials. Ara bé, tingueu cura a l'hora de comprovar que realitzeu bé les operacions corresponents a les opcions 6 i 7 del menú, de trobar



# Estructura de Dades: Pràctica 3

l'identificador més gran (opció 6) i de l'identificador més petit que no estigui assignat a cap director existent a Mubiesflix (opció 7). Amb les dades de `pelis-cas_de_prova.txt`, us haurien de donar:

```
Largest director id: 132306
Smallest not taken director id: 3
```

## Base de dades gran

Per mesurar els temps, podeu fer servir l'alternativa que vulgueu. Per tenir precisió inferior a segons, per exemple, mil·lèsimes de segon (ms), una bona opció pot ser la llibreria **std::chrono**:

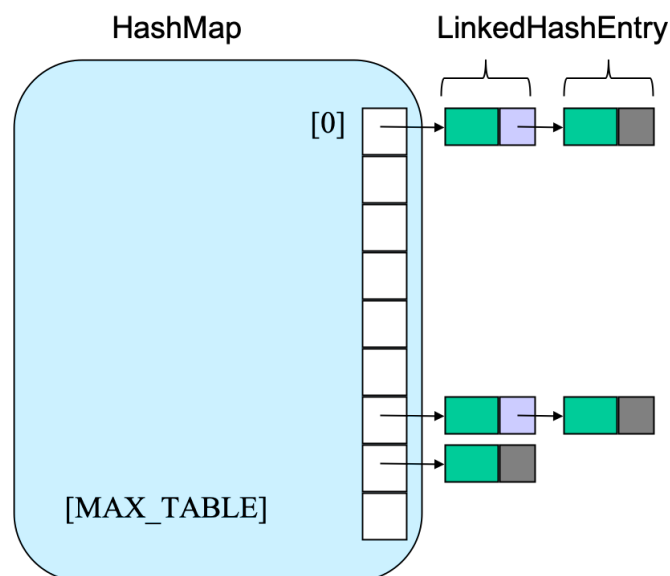
```
#include <chrono>
#include <iostream>
using namespace std;

chrono::steady_clock::time_point begin = chrono::steady_clock::now();
// Aquí el vostre codi del que en voleu mesurar el temps d'execució
chrono::steady_clock::time_point end = chrono::steady_clock::now();

cout << "Temps transcorregut: " << chrono::duration_cast<chrono::seconds>(end -
begin).count() << " s." << endl;
```

## 3. Taula de Hash Oberta

Aquesta part de la pràctica està centrada en la implementació d'una taula de hash oberta, basada en encadenament amb llistes enllaçades. Per fer-ho, disposareu d'una estructura bàsica amb dos TADs:



**LinkedListEntry:** representa els nodes de cada llista.

**HashMap:** conté la taula de hash i els mètodes d'inserció, consulta i gestió.

# Estructura de Dades: Pràctica 3

Heu d'acabar d'implementar els TADs que representen una taula de hash oberta amb encadenament:

A continuació teniu la definició de: *LinkedHashEntry<Key, Value>*

Operació	Definició
<b>constructor</b>	Inicialitza la clau i el punter al següent node a nullptr.
<b>destructor</b>	Allibera recursivament la memòria dels nodes encadenats
<b>getKey</b>	Retorna la clau del node.
<b>getValue</b>	Retorna el vector de valors associats.
<b>setValue</b>	Afegeix un nou valor al vector.
<b>getNext / setNext</b>	Getters i setters per enllaçar els nodes.
<b>toString</b>	Imprimeix la clau i tots els valors del node.

**LinkedHashEntry.h**

```
template <class Key, class Value>
class LinkedHashEntry {
private:
    Key key;
    vector<Value> values;
    LinkedHashEntry* next;
public:
    LinkedHashEntry(Key key);
    ~LinkedHashEntry();
    Key getKey();
    vector<Value>& getValue();
    void setValue(const Value& value);
    LinkedHashEntry *getNext();
    void setNext(LinkedHashEntry *next);
    void toString() const;
};
```

A continuació teniu la definició de: *HashMap<Key, Value>*

Operació	Definició
<b>Constructor / Destructor</b>	Inicialitza HashMap o destrueix el contingut de totes les cel·les de la taula.
<b>put</b>	Afegeix una entrada nova a la taula. Si hi ha col·lisió, afegeix el node al final de la llista.

# Estructura de Dades: Pràctica 3

<code>get</code>	Retorna si una clau es troba a la taula i imprimeix el seu contingut.
<code>size</code>	Retorna la mida de la taula
<code>cells</code>	Retorna el nombre de cel·les ocupades al HashMap
<code>colisions</code>	Retorna el nombre màxim de col·lisions que s'han produït
<code>getHashCode</code>	Retorna l'índex dins de la taula per una clau donada.
<code>getCell()</code>	Donat una posició 'i' a la taula, retorna el primer node de la cel·la i.
<code>getPosition()</code>	Donada una 'key', retorna el node (LinkedHashEntry) de la key si existeix.
<code>print</code>	Imprimeix tot el contingut de la taula.

## HashMap.h

```
template <class Key, class Value>
class HashMap {
public:
    const static int MAX_TABLE = ??; // Definir a ?? el valor que necessiteu
    HashMap();
    virtual ~HashMap();
    int getHashCode (int key) const;
    void put(const Key& key, const Value& value);
    void print();
    const bool get(const Key& key) const;
    int size() const;
    LinkedHashEntry <Key,Value>* getPosition(const Key& element);
    LinkedHashEntry<Key,Value>* getCell(int index) const;

private:
    LinkedHashEntry<Key,Value>* arrayElems[MAX_TABLE] = {}; // taula de hash
    int mida; // Nombre d'elements dins del HashMap
    int celles; // Nombre de cel·les ocupades del HashMap
    int colisioMax; // Profunditat màxima d'una cel·la a tot el HashMap
};
```

## Cas de prova

A main.cpp heu de validar el comportament del TAD HashMap amb el següent escenari:

### test

```
void test_hashmap(){
    HashMap<int,int> map;
    int testArray[] = { 6, 21, 24, 45, 13, 20, 25 };
    std::vector<int> values;
```

# Estructura de Dades: Pràctica 3

```
for (int i = 0; i < 7; i++) {
    cout << "Inserta al mapa " << testArray[i] << endl;
    values.push_back(i);
    for (int v : values) {
        map.put(testArray[i], v);
    }
}
map.print();

cout << "get(0)" << endl;
cout << map.get(0) << endl;
cout << "get(6)" << endl;
cout << map.get(6) << endl;

cout << "Size of the map: " << map.size() << endl;
cout << "Cells on the map " << map.cells() << " elements " << endl;
cout << "MaxColisions on the map " << map.colisions() << " elements " << endl;
}
```

Heu de comparar la sortida obtinguda amb el comportament esperat: que es mostrin les col·lisions, les claus repetides tinguin múltiples valors, i el nombre total d'elements i col·lisions sigui coherent.

## 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.

# Estructura de Dades: Pràctica 3

**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**