

Estructura de Dades: Pràctica 3

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

Introducció

Objectius: Familiaritzar-se amb les estructures no lineals, concretament, amb la implementació dels arbres binaris de cerca i els heaps, analitzant les seves característiques i diferències.

Temes de teoria relacionats amb la pràctica: Tema 4 Estructures no lineals: Arbres, Tema 5: Heaps

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
...
```

Noteu que les dades, per cada fila, hi consten: (1) el ID de la pel·lícula guardat com a `int`, (2) el títol i l'any guardat a `string`, (3) la puntuació de la pel·lícula (que li direm rating) que es guarda com a `float`.

Amb una estructura d'arbre binari de cerca, haureu d'oferir diferents consultes, entre d'altres, 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.

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.
- 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.

Estructura de Dades: Pràctica 3

1. Arbre binari de cerca

Implementareu, el TAD **ArbreBinari** amb templates que representi l'arbre binari de cerca amb una representació de nodes encadenats. Els nodes de l'arbre seran cadascun un TAD **NodeBinari**, que també haureu d'implementar i que representarà una clau i un element. 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:

ArbreBinari.h

```
template <class Clau, class Valor>
class ArbreBinari {
public:
    ArbreBinari();
    ArbreBinari(const ArbreBinari<Clau, Valor>& orig);
    virtual ~ArbreBinari();

    bool isEmpty() const;
    int height() const;

    NodeBinari<Clau,Valor>* insert(const Clau& clau, const Valor& value);
    const Valor& valueOf(const Clau& clau) const;
    void imprimirPreordre(const NodeBinari<Clau,Valor>* n = nullptr) const;
    void imprimirInordre(const NodeBinari<Clau,Valor>* n = nullptr) const;
    void imprimirPostordre(const NodeBinari<Clau,Valor>* n = nullptr) const;

    vector<NodeBinari<Clau,Valor>*> obteValorsRang(int k1, int k2) const;
    void imprimirCami(Clau key) const;
    void eliminaMinim();
protected:
    NodeBinari<Clau,Valor>* root;
    NodeBinari<Clau,Valor>* search(const Clau& k) const;

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

NodeBinari.h

```
template <class Clau, class Valor >
class NodeBinari {
public:
    NodeBinari(const Clau& key, const Valor& v);
    NodeBinari(const NodeBinari<Clau,Valor >& orig);
    virtual ~NodeBinari();

    /* Modificadors */
    // declareu aquí els modificadors (setters) dels atributs que necessiteu

    /* Consultors */
    const Clau& getKey() const;
    const Valor& getValue() const;
```

Estructura de Dades: Pràctica 3

```
// declareu aquí els consultors (getters) dels atributs que necessiteu

/* Operacions */
bool isRoot() const;
bool hasLeft() const;
bool hasRight() const;
bool isExternal() const;
void insertValue(const Valor & v);
int height() const;
bool operator==(const NodeBinari<Clau, Valor >& node) const;

private:
    Clau key;
    Valor value;
    // afegiu aquí els atributs que falten
};
```

Pas 1 Implementar el node de l'arbre binari de cerca

A continuació teniu la descripció de les operacions del TAD **NodeBinari** (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>)
getValue	Retorna el valor del node (atribut <code>value</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 el valor.
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

Pas 2 Implementar l'arbre binari de cerca

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

Estructura de Dades: Pràctica 3

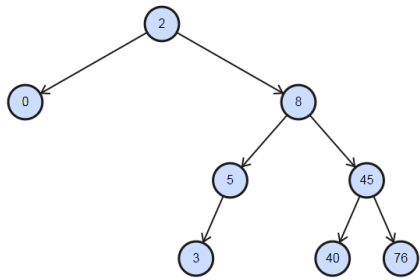
Operació	Definició
<code>constructor</code>	Construeix l'arbre buit.
<code>constructor còpia</code>	Construeix una còpia de l'arbre partir de l'arbre original rebut per paràmetre
<code>destructor</code>	Destruïx els nodes de l'arbre binari, començant pel root. <i>Atenció a la gestió de memòria dinàmica</i>
<code>isEmpty</code>	Retorna cert si l'arbre binari està buit, fals en cas contrari
<code>height</code>	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.
<code>insert</code>	Afegeix un nou node a l'arbre binari de cerca
<code>valueOf</code>	Retorna el valor (atribut <code>value</code>) d'un node de l'arbre amb una certa <code>key</code> passada per paràmetre
<code>imprimirPreordre</code>	Mostra per consola les claus seguint un recorregut en preordre
<code>imprimirInordre</code>	Mostra per consola les claus seguint un recorregut en inordre
<code>imprimirPostordre</code>	Mostra per consola les claus seguint un recorregut en postordre
<code>obteValorsRang</code>	Retorna un vector amb tots els nodes que són igual o estan entre els dos rangs donats. Si no hi ha elements, el vector retornarà buit. Aquesta funció s'ha d'implementar de forma RECURSIVA
<code>imprimirCami</code>	Mostra per consola totes les claus des de l'arrel fins a arribar a la clau donada per paràmetre. En el cas que no existeixi la clau, el camí fins a on s'ha arribat per cercar la clau. Aquesta funció s'ha d'implementar de forma RECURSIVA
<code>eliminaMinim</code>	Elimina el node que conté la clau més petita a tot l'arbre. Si l'arbre està buit, no fa res. Aquesta funció s'ha d'implementar de forma ITERATIVA
<code>search</code> <code>(protected)</code>	Troba un element (indexat per la clau) de l'arbre binari de cerca i retorna el node si el troba, en cas contrari retorna <code>nullptr</code> . 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 plantegem 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`).

Estructura de Dades: Pràctica 3

Cas de Prova

Pas	Entrada	Sortida
1	Crea un arbre binari de cerca (anomenat <code>tree1</code>) que emmagatzemi la clau entera i els valors enters	Arbre binari creat Arbre binari creat
2	<p>Inserir a l'arbre (<code>tree1</code>) les claus d'un array anomenat <code>testKeys</code> i les seves values (en l'array <code>testValues</code>) que contindrà:</p> <pre>int testKeys[] = {2, 0, 8, 45, 76, 5, 3, 40}; int testValues[] = {5, 5, 1, 88, 99, 12, 9, 11};</pre> <p>L'arbre <code>tree1</code> amb les claus quedarà:</p>  <pre> graph TD 2((2)) --> 0((0)) 2 --> 8((8)) 8 --> 5((5)) 8 --> 45((45)) 5 --> 3((3)) 45 --> 40((40)) 45 --> 76((76)) </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 (<code>tree1</code>) per pantalla	PreOrdre = {2, 0, 8, 5, 3, 45, 40, 76}
4	Mostrar en inordre l'arbre (<code>tree1</code>) per pantalla	InOrdre = {0, 2, 3, 5, 8, 40, 45, 76}
5	Mostrar en postordre l'arbre (<code>tree1</code>) per pantalla	PostOrdre = {0, 3, 5, 40, 76, 45, 8, 2}
6	Cridar a la funció <code>tree1.obteValorsRang(5, 45)</code>	ObteValorsRang={5,8,40,45}
7	<p>Fer una còpia de l'arbre <code>tree1</code>, anomenada <code>tree2</code></p> <p>Mostrar el postordre de <code>tree2</code></p>	<p>Arbre binari copiat</p> <p>PostOrdre = {0, 3, 5, 40, 76, 45, 8, 2}</p>
8	Cridar a la funció <code>tree2.imprimirCami(40)</code>	ImprimirCami={2,8,45,40}
9	<p>Eliminar el mínim i fer l'inordre de l'arbre</p> <pre>tree2.eliminaMinim(); tree2.imprimirInordre();</pre>	InOrder = {2, 3, 5, 8, 40, 45, 76}
10	Eliminar l'arbre <code>tree1</code>	<p>Destruint arbre binari</p> <p>Eliminant NodeBinari 2</p> <p>Eliminant NodeBinari 0</p> <p>Eliminant NodeBinari 8</p> <p>Eliminant NodeBinari 5</p> <p>Eliminant NodeBinari 3</p> <p>Eliminant NodeBinari 45</p> <p>...</p> <p>Arbre binari destruït</p>

Estructura de Dades: Pràctica 3

Exercici 2. Cercador de pel·lícules amb un arbre 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.
- **Constructor còpia:** construeix una Movie a partir d'una altra objecte de tipus Movie.
- **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, separat per :: cada camp a dins l'string.
- **Funció print:** imprimirà la Movie en format (atribut_1:: :: atribut_N).

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

Escriviu l'especificació d'una classe anomenada **CercadorMoviesAB** 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. `insertarMovie(movieID, title, rating)`: Aquest mètode rep les dades d'una pel·lícula i fa la inserció a l'estructura de dades corresponent.
3. `mostrarMovie(movieID)`: Aquest mètode rep un identificador de pel·lícula i retorna un sol string amb les dades associades a la Movie.
4. `buscarMovie(movieID)`: Aquest mètode no ha de retornar un string, retorna l'objecte 'movie'.
5. `buscarRatingMovie(movieID)`: Aquest mètode ha de retornar el rating de la pel·lícula.
6. `eliminarMinimaMovie(n)`: Aquest mètode ha d'esborrar les n pel·lícules que tinguin l'identificador més petit a tot l'arbre.

Pas 2. Implementació del Cercador de pel·lícules amb un arbre

Implementeu l'especificació anterior, **CercadorMoviesAB**, amb un **ArbreBinari**. En aquest arbre els **NodeBinari** 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. Podeu donar per suposat que la *key* serà sempre un ID de pel·lícula de tipus int i la *value* serà l'objecte Movie.

Estructura de Dades: Pràctica 3

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

```
class CercadorMoviesAB: protected ArbreBinari<int, Movie>{
public:
    CercadorMoviesAB();
    CercadorMoviesAB(const CercadorMoviesAB & orig);
    virtual ~CercadorMoviesAB();

    void appendMovies(string filename);
    void insertarMovie(int movieID, string title, float rating);
    string mostrarMovie(int movieID);
    Movie buscarMovie(int movieID);
    float buscarRatingMovie(int movieID);
    void eliminarMinimaMovie(int n);
    int height() const;
private:
    // Aquí posareu tots els mètodes auxiliars
};
```

Pas 3. Programació d'un main amb menú interactiu

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 **CercadorMoviesAB**.
 - Inicialitzar el **CercadorMoviesAB** a partir del contingut del fitxer.
 - Mostrar el temps de creació del **CercadorMoviesAB**.
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 *cercaPelicles.txt* que us proporcionem i per a cada movieID contingut a directori, fer una cerca en l'arbre emmagatzemat a CercadorMoviesAB. Avaluar el temps de cerca de tots els elements de *cercaPelicles.txt* i comptar el nombre d'elements que estan en el CercadorMoviesAB. Mostrar les dues dades per pantalla.
4. Visualitzar per pantalla la profunditat de l'arbre.
5. Esborrar les n pel·lícules que tenen el id més petit a l'arbre. El valor de n es demana a l'usuari.
6. Sortir del programa

Implementeu aquestes funcionalitats en forma d'un menú al main, a partir d'una funció `mainArbreBinari`, que permeti realitzar les 5 accions descrites anteriorment i l'opció 6 serà sortir del menú.

Atenció: En aquest exercici sols es demana fer amb templates els TAD `NodeBinari` i `ArbreBinari`. El TAD `Movie` o `CercadorMoviesAB` no cal fer-los amb templates.

Estructura de Dades: Pràctica 3

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;
```

NOTA: El cas de prova de l'exercici 1, es posarà com la primera crida del main, tot just abans aparegui el menú interactiu.

Exercici 3: Heap

Implementeu el TAD **MinHeap** amb **templates** que representi un heap. La funcionalitat d'aquest TAD ha de ser exactament la mateixa que el TAD **ArbreBinari** de l'Exercici 1. Considereu si podeu re-utilitzar directament el TAD **NodeBinari**, o si pel contrari, trobeu que requereix també modificacions. Al node del TAD **MinHeap**, li posarem de nom **NodeHeap**.

Expliqueu a `MinHeap.h` les similituds i diferències en la implementació d'aquest TAD respecte el `ArbreBinari.h`. Recordeu detallar al `.h` quin és el cost computacional teòric en el pitjor dels casos de cadascuna de les operacions del TAD. A continuació teniu l'especificació del TAD.

```
template<class Clau, class Valor>
class MinHeap {
public:
    MinHeap(); // constructor
    MinHeap<Clau,Valor> (const MinHeap<Clau,Valor>&); // constructor còpia
    int tamany(); // retorna el nombre d'elements que hi ha al heap
    bool esBuit(); // retorna cert si és buit, fals en cas contrari
    void inserir(const Clau& clau, const Valor& valor);
    Clau minim(); // retorna la clau mínima
    Valor valorMinim(); // retorna el valor de la clau mínima
    void eliminaMinim(); // elimina el node amb valor mínim
    void imprimir(); // Imprimeix per pantalla tot el Heap, una línia per a cada nivell
de l'arbre
    Valor cerca(const Clau& clau); // cerca al heap una clau donada
    int altura(); // retorna l'alçada del heap

private:
    vector< NodeHeap<Clau, Valor> > data;
    int mida;
    // aquí sota definiu les funcions auxiliars
};
```


Estructura de Dades: Pràctica 3

Cas de prova

```
void casDeProvaExercici3() {
    std::cout << " ----- cas de prova exercici 3 ----- " << std::endl;
    MinHeap<int, int> heap1;

    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++) {
        heap1.inserir(testKeys[i], testValues[i]);
    }
    cout << "heap1 ={ ";
    heap1.imprimir();
    cout << "}" << endl;

    cout<<"Mida Heap recent creat: "<<heap1.tamany()<<endl;
    cout<<"Alçada Heap recent creat: "<<heap1.altura()<<endl;
    cout<<"Heap recent creat. is esBuit? (0/1): "<<heap1.esBuit()<<endl;
    cout << "Clau minima " << heap1.minim() << endl;

    cout << "EliminaMinim" << endl;
    heap1.eliminaMinim();
    cout<<"Heap despres eliminar 1. Print: "<<endl;
    heap1.imprimir();
    cout<<"Mida Heap recent creat: "<<heap1.tamany()<<endl;
    cout<<"Alçada Heap recent creat: "<<heap1.altura()<<endl;

    heap1.eliminaMinim();
    cout<<"Heap despres eliminar 2. Print: "<<endl;
    heap1.imprimir();
    cout<<"Mida Heap recent creat: "<<heap1.tamany()<<endl;
    cout<<"Alçada Heap recent creat: "<<heap1.altura()<<endl;
}
```

Exercici 4: Cercador de pel·lícules amb un heap

La classe **CercadorMoviesAB** usa un **ArbreBinari**. Ara fareu el **CercadorMoviesHeap** que usara un **Heap**. Prepareu al main dues funcions, una primera **mainArbreBinari()** i una funció anomenada **mainHeap()**, cadascuna farà el menú amb una estructura de dades diferent, l'arbre binari o el heap. És important que mantingueu el nom de cada funció de 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:

Estructura de Dades: Pràctica 3

- Crear un objecte **CercadorMoviesHeap**.
 - Inicialitzar el **CercadorMoviesHeap** a partir del contingut del fitxer.
 - Mostrar el temps de creació del **CercadorMoviesHeap**.
2. Mostrar el heap per nivells.
 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 CercadorMoviesHeap. Avaluar el temps de cerca de tots els elements de *cercaPelicules.txt* i comptar el nombre d'elements que estan en el CercadorMoviesHeap. Mostrar les dues dades per pantalla.
 4. Visualitzar per pantalla la profunditat del heap.
 5. Esborrar les n pel·lícules que tenen el id més petit a l'arbre. El valor de n es demana a l'usuari. Abans d'esborrar cada pel·lícula, es busca i es mostra per pantalla totes les dades de la pel·lícula.
 6. Sortir del programa

Exercici 5: Avaluació d'estructures

Feu una avaluació del rendiment experimentat de les dues implementacions anteriors (arbres de cerca binària i heaps), i raoneu les diferències: compteu el temps de generació de l'estructura per dos fitxers de diferent grandària i compteu el temps d'accés (cerca) del fitxer de consulta en les estructures generades a partir dels dos fitxers que trobareu al campus virtual: un fitxer gran (*movie_rating.txt*) i un de petit (*movie_rating_small.txt*).

Feu les proves en el mateix ordinador i en condicions similars. Raoneu els resultats de temps obtinguts i poseu una taula com la següent a dalt de tot del main.cpp. Indiqueu en cada casella el temps de cada operació en cadascun dels arbres.

Operació	ArbreBinari	MinHeap
Inserció arbre petit <i>movie_rating_small.txt</i>		
Inserció arbre gran <i>movie_rating.txt</i>		
Cerca arbre petit <i>cercaPelicules.txt</i>		
Cerca arbre gran <i>cercaPelicules.txt</i>		

Estructura de Dades: Pràctica 3

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: `movie_rating.txt`, `movie_rating_small.txt` i el fitxer de `cercaPelicules.txt`.

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 19 d'abril al 21 de maig 2023. Per aquesta pràctica els professors proposen la següent planificació:

- **Setmana 1** (*Classe de Laboratori 8*)
 - Implementació de la classe `Movie`
 - Fer una funció que llegeixi el fitxer i el guardi cada línia a un objecte `Movie`
- **Setmana 2** (*Classe de Laboratori 9*)
 - Exercici 1: Implementació de l'**ArbreBinari** i el **NodeBinari**, el main, i comentar el codi
 - Fer el cas de prova inicial amb enters
- **Setmana 3** (*Classe de Laboratori 10*)
 - Exercici 2: Implementació de la classe **CercadorMoviesAB** i del main.cpp, i comentar el codi
- **Setmana 4** (*Classe de Laboratori 11*)
 - Exercici 3: Implementació del **MinHeap**, el main, i comentar el codi
- **Setmana 5** (*Classe de Laboratori 12*)
 - Exercicis 4 i 5: **CercadorMoviesHeap**, fer les proves de l'exercici 5 i comentar el codi
- **Setmana 6** (*Classe de Laboratori 13*)
 - **PROVA DE LA PRÀCTICA 3 al laboratori.**

Recordeu que la setmana 5 s'ha de lliurar la totalitat de la pràctica (tots els exercicis al ZIP).

Lliurament: dia 21 de maig de 2023