

Estructura de Dades: Pràctica 3

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

Introducció

Objectius: Familiaritzar-se amb les estructures no lineals, concretament, amb les diferents implementacions dels arbres binaris de cerca, analitzant les seves característiques i diferències.

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

Enunciat

Un corredor de borsa online ens demana implementar certes funcions per l'anàlisi de les transaccions realitzades per part dels seus clients en un instant o interval de temps qualsevol i, a partir de les quantitats de les transaccions, el càlcul de les comissions generades.

El corredor (o broker) ens ha facilitat un fitxer on hi té emmagatzemades totes les transaccions (una per línia) dels seus clients l'any 2020. Per cada transacció, hi consten: **(1)** l'instant de temps, data i hora, de la transacció (`time` en format "YYYY-MM-DD HH:mm") guardat com string, **(2)** l'identificador intern dels seu client (`client_id`) guardat com integer, i **(3)** la quantia de la transacció (`amount`) en euros guardada com a float, que serà negativa si es tracta d'una compra d'accions per part del seu client o positiva en cas de tractar-se d'una venda.

Aquest és un exemple de fitxer de transaccions (inclou una capçalera):

```
time,client_id,amount
2020-01-01 09:23,11712,107.78
2020-01-01 09:29,13708,5121.25
2020-01-01 12:48,24190,1575.12
2020-01-01 13:47,25088,-149.52
2020-01-01 13:56,5395,2268.61
2020-01-01 14:23,6094,-1068.34
2020-01-01 14:45,17893,262.73
2020-01-01 14:52,23209,-5823.66
2020-01-01 15:16,24757,-1489.3
2020-01-01 15:24,6049,-1506.71
2020-01-02 09:27,2744,-11161.46
2020-01-02 09:36,27769,-227.6
2020-01-02 09:42,14245,2881.39
2020-01-02 10:07,3736,-487.7
2020-01-02 10:38,25209,138.51
```

Amb una estructura d'arbre binari de cerca, haureu d'oferir al corredor serveis per, entre d'altres coses, calcular la comissió total generada per les compres (suma dels negatius) i les vendes (suma dels positius) d'accions en un instant de temps concret. Tingueu en compte que el corredor cobra una comissió percentual, que pot ser diferent per a les compres i les vendes. Per exemple, si es demana la comissió que suposen les transaccions de l'instant amb data i hora "2020-01-01 15:16" i els percentatges de comissió sobre compra i venda són del 2% i 3% respectivament, el sistema haurà de retornar la comissió de $0.02 * \text{abs}(-1489.3) + 0.03 * 0 = 29.78$. Penseu que en un determinat instant poden haver-se produït una o més transaccions

Estructura de Dades: Pràctica 3

i que, per tant, la comissió s'aplicarà sobre una sèrie de compres i de vendes associades a l'instant. El mateix passarà si se'ns demana realitzar el càlcul de la comissió sobre un interval de temps. Per exemple, les comissions generades entre "2020-01-01 12:00" i "2020-01-01 14:00" serien totes les que es varen donar en els instants de temps "2020-01-01 12:48", "2020-01-01 13:47" i "2020-01-01 13:56", és a dir, el guany pel cobrament de les comissions serà $0.02 * \text{abs}(-149.52) + 0.03 * (1575.12 + 2268.61) = 118.30$.

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.
 - 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 és opcional**, però altament recomanable.
- 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. Arbore binari de cerca

Implementareu, el TAD **BinarySearchTree** que representi l'arbre binari de cerca amb nodes encadenats. Els nodes de l'arbre seran cadascun un TAD **BinaryTreeNode**, que també haureu d'implementar i que representarà una clau i un vector 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:

```
BinarySearchTree.h
template <class K, class V>
class BinarySearchTree {
public:
    BinarySearchTree();
    BinarySearchTree(const BinarySearchTree<K, V>& orig);
    virtual ~BinarySearchTree();

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

    BinaryTreeNode<K,V>* add(const K& k, const V& value);
    bool has(const K& k) const;
    const vector<V>& valuesOf(const K& k) const;

    void showKeysPreorder(const BinaryTreeNode<K,V>* n = nullptr) const;
    void showKeysInorder(const BinaryTreeNode<K,V>* n = nullptr) const;
    void showKeysPostorder(const BinaryTreeNode<K,V>* n = nullptr) const;

    bool equals(const BinarySearchTree<K, V>& other) const;
    const vector<BinaryTreeNode<K, V>*>& getLeafs() const;

protected:
    BinaryTreeNode<K,V>* p_root;
    BinaryTreeNode<K,V>* find(const K& k) const;
```

Estructura de Dades: Pràctica 3

```
BinaryTreeNode.h
template <class K, class V>
class BinaryTreeNode {
public:
    BinaryTreeNode(const K& key);
    BinaryTreeNode(const BinaryTreeNode<K,V>& orig);
    virtual ~BinaryTreeNode();

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

    /* Consultors */
    const K& getKey() const;
    const vector<V>& getValues() const;
    // Declareu-hi aquí els consultors (getters) dels atributs que manquen

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

    void addValue(const V& v);
    int depth() const;
    int height() const; // uses auxiliary attribute
    bool operator==(const BinaryTreeNode<K,V>& node) const;

private:
    K key;
    vector<V> 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 **BinaryTreeNode** (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
isLeaf	Retorna cert si el node és una fulla de l'arbre binari de cerca, fals altrament
addValue	Afegeix un valor a la llista 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

Estructura de Dades: Pràctica 3

<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 **BinarySearchTree** són les següents:

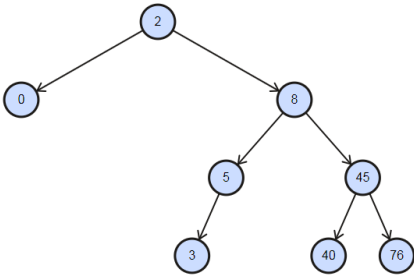
Operació	Definició
<code>constructor</code>	Construeix l'arbre buit.
<code>constructor còpia</code>	Construeix una còpia de l'arbre partir del node original rebut per paràmetre
<code>destructor</code>	Destruïx els nodes de l'arbre binari, començant pel root. <i>Atenció a la gestió de memòria dinàmica</i>
<code>isEmpty</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. Aquesta funció recorre els nodes per calcular quants nodes té 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. Aquesta funció s'ha d'implementar de forma RECURSIVA
<code>add</code>	Afegeix un nou node a l'arbre binari de cerca
<code>has</code>	Retorna cert si l'arbre binari de cerca té l'element indexat amb una certa clau, fals en cas contrari. Aquesta funció s'ha d'implementar de forma RECURSIVA
<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>showPreOrder</code>	Mostra per consola les claus i el contingut de les claus dels nodes seguint un recorregut en preordre
<code>showInOrder</code>	Mostra per consola les claus i el contingut les claus dels nodes seguint un recorregut en inordre
<code>showPostOrder</code>	Mostra per consola les claus i el contingut les claus dels nodes seguint un recorregut en postordre
<code>equals</code>	Retorna cert si l'arbre binari intern és igual a l'arbre binari que es passa per paràmetre, fals en cas contrari. Dos arbres són iguals si tots els seus nodes ho són
<code>getLeafs</code>	Retorna un vector amb tots els nodes fulla de l'arbre.
<code>find</code> (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.

Estructura de Dades: Pràctica 3

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`).

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> 	<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	PreOrder = {2, 0, 8, 5, 3, 45, 40, 76}
4	Mostrar en inordre l'arbre (<code>tree1</code>) per pantalla	InOrder = {0, 2, 3, 5, 8, 40, 45, 76}
5	Mostrar en postordre l'arbre (<code>tree1</code>) per pantalla	PostOrder = {0, 3, 5, 40, 76, 45, 8, 2}
6	Fer una còpia de l'arbre <code>tree1</code> , anomenada <code>tree2</code>	Arbre binari copiat
7	Cridar a la funció <code>tree1.equals(tree2)</code>	Cert
8	Afegir a l'arbre <code>tree1</code> l'element amb clau 1 i valor 10	Inserta a l'arbre l'element 1
9	Cridar a la funció <code>tree2.equals(tree1)</code>	Fals
10	Eliminar l'arbre <code>tree1</code>	<p>Destruint arbre binari Eliminant BinaryTreeNode 2 Eliminant BinaryTreeNode 1 Eliminant BinaryTreeNode 0 Eliminant BinaryTreeNode 8 Eliminant BinaryTreeNode 5 Eliminant BinaryTreeNode 3 Eliminant BinaryTreeNode 45 ... Arbre binari destruït</p>

Estructura de Dades: Pràctica 3

2. Gestor de transaccions amb un arbre binari de cerca

Pas 1 Especificació de Transaction

Implementeu el TAD **Transaction** que representi i emmagatzemi la informació d'una transacció (els 3 valors de cada línia del fitxer de transaccions). En aquest exercici, **Transaction** serà el tipus de dades en el vector de valors dels nodes, és a dir, cada node de l'arbre binari de cerca tindrà un vector de **Transaction**. Haureu de definir-ne:

Operació	Definició
Constructor i constructor còpia	Construeix una transacció amb totes les seves dades: instant de temps de la transacció (<code>string</code> en format YYYY-MM-DD HH:mm), ID del client (<code>int</code>), i quantitat de la transacció (<code>float</code>).
funcions consultores	Una o més funcions per consultar cada una de les dades associades a la transacció
funcions modificadores	Una o més funcions per modificar cada una de les dades associades a la transacció
print	Imprimirà la transacció en format: (<code>atribut_1, ..., atribut_N</code>)

Pas 2 Especificació de Gestor de Transaccions (TransactionManager)

Implementeu la classe **TransactionManager** que hereti de la classe **BinarySearchTree<string, Transaction>** en cas que hagueu implementat l'arbre de cerca binària amb templates.

```
TransactionManager.h
(BinarySearchTree implementat amb templates)
class TransactionManager : protected BinarySearchTree<string, Transaction> {
    ...
}
```

En cas contrari, copieu el codi de l'Exercici 1 i feu-hi NOMÉS els canvis estrictament necessaris, tenint en compte que les claus dels nodes passen de ser `int` a `string` i que el valors del vector de cadascun dels nodes de l'arbre passen de ser `int` a **Transaction**. Finalment, feu que **TransactionManager** hereti simplement de **BinarySearchTree**.

```
TransactionManager.h
(BinarySearchTree implementat sense templates)
class TransactionManager : protected BinarySearchTree {
    ...
}
```

En qualsevol cas, és important que l'herència sigui amb visibilitat `protected`. Això evitarà que exposeu els mètodes de la classe mare -- és a dir, del TAD **BinarySearchTree** -- des d'aquesta nova classe **TransactionManager**.

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

Estructura de Dades: Pràctica 3

```
class TransactionManager : protected BinarySearchTree<string, Transaction> {
public:
    TransactionManager(float buyingFee = 0.02, float sellingFee = 0.03);
    TransactionManager(string file_path, float buyingFee = 0.02, float sellingFee = 0.03);
    virtual ~TransactionManager();

    void loadFromFile(string file_path);
    void showAll() const;
    void showAllReverse() const;
    void showOldest() const;
    void showNewest() const;

    float feesInTotal() const;
    float feesSinceTime(string date) const;
    float feesInTimeInterval(pair<string, string> interval) const;

private:
    float sellingFee;
    float buyingFee;
```

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

Operació	Definició
Constructor i constructor còpia	Constructor amb paràmetres per defecte, constructor amb paràmetre que rebi la ruta del fitxer de transaccions amb dades per carregar l'arbre (i resta de paràmetres per defecte) i constructor còpia Els paràmetres per defecte seran el percentatge de comissió de compra i de venda. Per defecte, 2% i 3% respectivament.
Destructor	Destructor de la classe
loadFromFile	Carrega un fitxer en l'arbre a partir de la ruta d'un fitxer de transaccions
showAll	Imprimeix les transaccions. Cada línia impresa tindrà l'instant de temps i la llista de transaccions realitzades en aquell instant (separades per comes). Les dates s'imprimiran en ordre temporal. Demanarà quantes dates imprimir, les imprimirà, i després demanarà quantes més se'n volen imprimir. Es repetirà el procés fins que l'usuari indiqui un número < 1
showAllReverse	Igual que showAll() però en ordre temporal invers.
showOldest	Imprimeix únicament les transaccions de l'instant de temps més antic
showNewest	Imprimeix únicament les transaccions de l'instant de temps més recent
feesInTotal	Retorna la quantitat de comissions guanyades pel broker aplicant les comissions a totes les transaccions del fitxer carregat
feesSinceTime	Retorna la comissió guanyada en totes les transaccions a partir d'un determinat instant de temps, aquest inclòs
feesInTimeInterval	Retorna la comissió guanyada en totes les transaccions en un interval de temps, incloent-hi els temps límits de l'interval. Per l'interval utilitzeu la classe std::pair (del paquet <utility>)

Estructura de Dades: Pràctica 3

Tingueu en compte que **TransactionManager** hereta del TAD **BinarySearchTree** 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 en el Gestor de Transaccions. Us proporcionarem, mitjançant el Campus Virtual, diversos fitxers `trading-*.txt` que haureu d'afegir al directori arrel del vostre projecte i com a "Resource Files" del vostre projecte NetBeans.
2. Mostrarà totes les transaccions ordenades temporalment, demanant primer quantes se'n desitgen mostrar de cop i, després d'haver-les mostrat, demanarà quantes més se'n volen mostrar. Si l'usuari no desitja seguir, entrarà per teclat un 0.
3. Mostrarà totes les transaccions en ordre temporal invers, de la mateixa manera que es fa a l'opció 2.
4. Mostrarà les transaccions del primer instant de temps del fitxer de transaccions carregat.
5. Mostrarà les transaccions del darrer instant de temps del fitxer de transaccions carregat.
6. Mostrarà la comissió recaptada amb totes les transaccions.
7. Mostrarà la comissió recaptada a partir d'una data específica.
8. Mostrarà la comissió recaptada entre dues dates. La interacció amb l'usuari serà semblant a la de l'opció 5, amb la diferència que se li demanaran les dues dates que delimiten l'interval temporal.
9. Mostrarà el balanç de totes les transaccions efectuades en les dates llistades en un fitxer `queries.txt`. Aquest fitxer no contindrà transaccions, sinó només dates que serviran de clau per anar a buscar les transaccions ja carregades al **TransactionManager**. Igual que en l'opció 1, us demanarem mesurar el temps que triga a processar-se la petició.
10. Sortir

Cas de prova

Us suggerim que carregueu el fitxer `transactions-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. Les operacions corresponents a les opcions 2-5 són trivials de comprovar fent un cop d'ull al fitxer. Si executeu la resta d'operacions per les opcions del menú 6-9, hauríeu d'obtenir una sortida semblant a aquesta:

```
feesInTotal: 808.948
feesSinceTime: 504.526      → paràmetre "2020-01-01 14:52"
feesInTimeInterval: 480.815 → paràmetre ival=pair<string,string>("2020-01-01 14:52");
Balanç total sobre queries: 808.948
```

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;
```


Estructura de Dades: Pràctica 3

3. Arbre binari auto-balancejat

Implementeu el TAD **AVLTree** que representi un arbre binari auto-balancejat. La funcionalitat d'aquest TAD ha de ser exactament la mateixa que el TAD **BinarySearchTree** de l'Exercici 1. Feu que aquesta nova classe hereti del TAD **BinarySearchTree** i sobrecarregueu les funcions que s'hagin de modificar i/o afegiu-n'hi de noves. Considereu si podeu re-utilitzar directament el TAD **BinarySearchNode**, o si pel contrari, trobeu que requereix també modificacions.

Repetiu els experiments duts a terme a l'Exercici 1. Expliqueu a `AVLTree.h` les similituds i diferències en la implementació d'aquest TAD respecte el `BinarySearchTree`. Recordeu detallar al `.h` quin és el cost computacional teòric en el pitjor dels casos de cadascuna de les operacions del TAD.

4. Gestor de Transaccions amb arbre binari auto-balancejat

Implementeu una nova classe **TransactionManagerAVL** heretant, aquest cop, del TAD **AVLTree**. Els mètodes d'aquesta nova classe seran exactament els mateixos que els del TAD **TransactionManagerAVL**. 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.

5. Avaluació d'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 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 (al campus virtual trobareu un fitxer gran i un de petit).

Feu les 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.

	BinarySearchTree	AVLTree
Inserció arbre petit (transactions-small.txt)		
Inserció arbre petit barrejat (transactions-small.shuffled.txt)		
Cerca arbre gran (transactions-large.txt)		
Cerca arbre gran (transactions-large.shuffled.txt)		

Estructura de Dades: Pràctica 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.
- No us oblideu d'afegir a la carpeta Resources els fitxers de les proves: transactions-*.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 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).

Planificació

Del 8 d'abril al 16 de maig 2021. Per aquesta pràctica els professors proposen la següent planificació:

- **Setmana 1** (*Classe de Laboratori 7*)
 - Implementació de les classes BinaryTreeNode, Transactions i comentar el codi
 - Fer una funció que llegeixi el fitxer i el guardi cada línia a un objecte Transactions
- **Setmana 2** (*Classe de Laboratori 8*)
 - Implementació del BinarySearchTree, el main, i comentar el codi
- **Setmana 3** (*Classe de Laboratori 9*)
 - Implementació de la classe TransactionsManager i comentar el codi
- **Setmana 4** (*Classe de Laboratori 10*)
 - Implementació del main de l'exercici 2, la part de l'exercici 5 sobre arbres binaris i comentar el codi
- **Setmana 5** (*Classe de Laboratori 11*)
 - Implementació de l'exercici 3 i comentar el codi
- **Setmana 6** (*Classe de Laboratori 12*)
 - Implementació de l'exercici 4, exercici 5 i comentar el codi

La setmana 2, 4 i 6 s'obrirà una tasca al campus virtual per disposar la feina feta fins aquell moment. **Recordeu que la setmana 6 s'ha de lliurar la totalitat de la pràctica (tots els exercicis al ZIP).**

Lliurament: dia 16 de maig de 2021