INF2010
Structures de données et algorithmes
TP03 : Arbres

Hiver 2019

Objectifs du TP:

- Implémenter un arbre binaire de recherche.
- Implémenter un arbre représentant la structure d'une compagnie.
- Utiliser un arbre de binaire de recherche comme structure de données dans un cas réel.

1 Arbre binaire de recherche

Un arbre binaire de recherche est un arbre qui respecte les deux propriétés suivantes:

- La valeur du fils gauche d'un nœud donné est inférieure **ou égale** à la valeur du nœud courant.
- La valeur du fils droit d'un nœud donné est supérieure à la valeur du nœud courant.

Les arbres binaires de recherches sont normalement composés de clef uniques. mais pour ce problème, nous acceptons les **doublons**. Comme mentionné, les doublons sont placés à gauche.

1.1 Implémenter les fonctions nécessaires pour réaliser un arbre binaire de recherche.

Laissez-vous guider par les « TODO » et les commentaires dans les fichiers *BinaryNode.java* et *BinarySearchTree.java*
Vos algorithmes doivent être optimaux, la complexité asymptotique sera testée.

- BinaryNode :
  - Data : représente la donnée du problème, elle doit implémenter l'interface Comparable pour pouvoir la placer les items aux bons endroits.
  - Insert : on place le nouvel item à gauche s'il est plus petit ou égal à l'item courant, sinon à gauche.
  - Contains : on veut savoir si l'item est égal à l'item courant ou a un de ses enfants.
  - GetHeight : on va savoir la hauteur d'un certain nœud de bas en haut, la racine tout seul à une hauteur de 0 par défaut.
  - FillListInOrder : on remplit une liste déjà créée en y ajoutant les éléments en ordre logique, donc gauche, centre et puis droite.
- BinarySearchTree :
  - Insert : on insère dans la racine
  - Contains : est-ce que la racine contient l'élément
  - GetHeight : qu'elle est la hauteur de la racine
  - GetItemsInOrder : Passe une liste a la racine pour que ses enfants la remplissent en ordre logique
  - ToStringInOrder : retourne le string des items de l'arbre en ordre logique sous le format « [1, 2, 3] », on note les [], les virgules et les espaces.

2 Arbre de compagnie

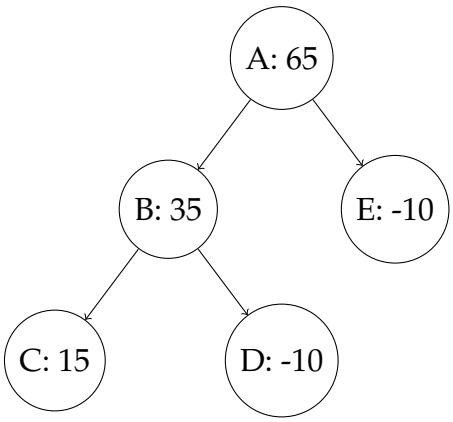
Plusieurs compagnies, quelles soient cotées en bourse ou non, font des acquisitions au cours de leur histoire. Certains achats se portent fructueux pour l'entreprise mère et d'autres sont simplement de mauvais choix... Pour simplifier le problème, on va seulement considérer le montant en banque d'une entreprise. Si la compagnie A d'une valeur de 10 achète la compagnie B d'une valeur de 5, A vaut maintenant 15. Il est important de considérer que certaines compagnies ont des dettes, donc des valeurs en banque négatives.
Par exemple, supposons les compagnies et leur montant en banque :

A : 40, B : 30, C : 15, D : -10, E : -10

Et une série d'achat :

- B achète C (B vaut 45)
- A achète E (A vaut 30)
- B achète D (B vaut 35)
- A achète B (A vaut 65)

Le résultat finalement:



2.1 Implémenter les fonctions nécessaires pour réaliser l'arbre de compagnie

Laissez-vous guider par les « TODO » et les commentaires dans les fichiers *CompanyNode.java* et *CompanyTree.java*

- CompanyNode
  - Money : représente le montant en banque d'une compagnie, ce montant peut être négatif, il est aussi la somme de tous les montants en banque de ses enfants et de lui-même.
  - Childs : représente les acquisitions d'une compagnie. Les acquisitions sont contenues dans le BST que vous avez fait précédemment. L'idée est de pouvoir les présentés en ordre de façon efficace plus tard.
  - WorstChild : représente la pire acquisition d'une compagnie, cetteacquisition peut avoir été effectué par elle-même ou par une de ses acquisitions. Donc ceci correspond au pire des childs et de tous les sous-childs et etc. (tout ce qui est en dessous du nœud courant). Dans l'exemple plus haut, la réponse serait D ou E pour A. Mais si par exemple, B vallait -30, ça aurait été B pour A.
  - Buy : on ajoute la compagnie à nos acquisitions et on update notre montant en banque.
  - FillStringBuilderInOrder : on rempli un SB en ordre de présentation, voir l'exemple dans le code. On veut afficher les enfants en ordre décroissant, donc de la plus grosses compagnies a la petite. On ajoute à chaque niveau le string « > ».
  - CompareTo : dans quel ordre veut-on afficher les données.
- CompanyTree :
  - Buy : la racine achète une compagnie
  - GetWorstChildMoney : on veut savoir le montant en banque de la pire acquisition, dans notre exemple, ce serait -10.
  - GetTreeInOrder : on retourne l'arbre qui respecte le format demandé, on passe un StringBuilder à la racine et un préfix.

3 Instructions pour la remise

Le travail doit être remis via Moodle :

- 12 Mars avant 23h55 pour le groupe 01
- 19 Mars avant 23h55 pour le groupe 02

Veuillez envoyer dans une archive de type $*$.zip qui portera le nom inf2010_lab2_MatriculeX_ MatriculeY (MatriculeX $<$ MatriculeY) :

- Vos fichiers .java

Les travaux en retard seront pénalisés de 20 % par jour de retard. Aucun travail ne sera accepté après 4 jours de retard.

3.1 Barème de correction

10 pts: Arbre Binaire de recherche

5 pts: Fonctionnalité

4 pts: Complexité

1 pts: Style

10 pts: Arbre de compagnie

5 pts: Fonctionnalité

4 pts: Complexité

1 pts: Style