INF2010 - Structures de données et algorithmes

Travail Pratique 3

Arbre binaire de recherche

Département de génie informatique et logiciel

École Polytechnique de Montréal

Automne 2019

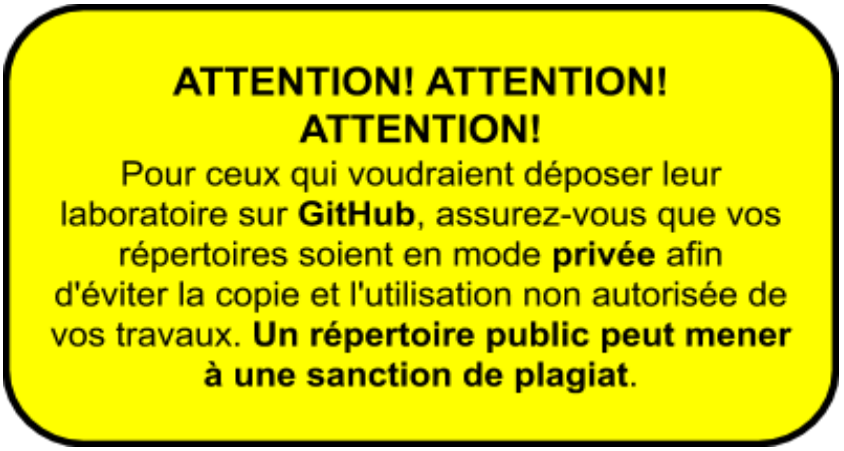
Objectifs

- Apprendre le fonctionnement d'un arbre binaire de recherche
- Comprendre la complexité en temps d'un arbre binaire de recherche

Pour ce laboratoire, il est recommandé d'utiliser l'IDE IntelliJ offert par JetBrains. Vous avez accès à la version complète (Ultimate) en tant qu'étudiant à Polytechnique Montréal. Il suffit de vous créer un compte étudiant en remplissant le formulaire au lien suivant:

https://www.jetbrains.com/shop/eform/students

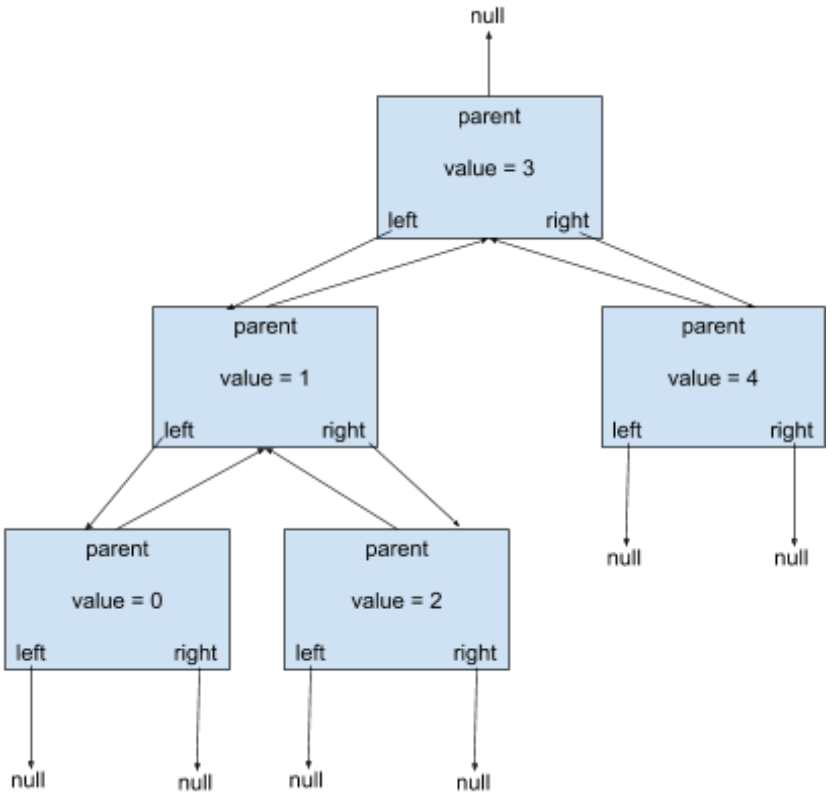
La correction du travail pratique sera partiellement réalisée par les tests unitaires implémentés dans les fichiers sources fournis. La qualité de votre code ainsi que la performance de celui-ci (complexité en temps) seront toutes deux évaluées par le correcteur. Un barème de correction est fourni à la fin de ce *.pdf.



Partie 1 : Implémentation d'un arbre AVL

Un arbre AVL est un arbre binaire de recherche équilibré. Celui-ci oblige que la différence entre la hauteur de l'arbre contenu du coté gauche et le niveau de l'arbre contenu du coté droit soit inférieur à 2. Si cette condition n'est pas respectée, il vous faut rééquilibrer l'arbre avec l'algorithme des arbres AVL. Dans le cas du retrait d'un élément, une démarche spécifique doit être utilisée. L'explication de ces algorithmes sont disponibles dans les diapositives Cours05 et Cours06.

L'arbre AVL sera composé d'un ensemble de *BinaryNode*s. Voici un exemple d'un arbre AVL utilisant ces *BinaryNode*s.



Pour bien implémenter l'arbre AVL, suivez les tests contenus dans AvlTreeTester.java. Aussi, n'oubliez pas que root est un cas d'exception pour la plupart des fonctions à implémenter.

Une note de 0 sera attribuée à cette partie si l'étudiant utilise un arbre binaire de recherche déjà implémentée provenant d'une librairie quelconque.

Astuce JAVA

```
public class AvlTree<ValueType extends Comparable<? super ValueType> > {
```

Décortiquons les parties intéressantes :

```
e<ValueType extends Comparable<? super ValueType> >
```
: Type générique nommé *ValueType* associé à la classe *AvlTree*

Un type générique permet à une classe de fonctionner avec n'importe quel type. Dans notre situation, on restreint les types qui seront acceptés dans notre classe.

```
extends Comparable<? super ValueType>
```
: Restriction sur le type générique

ValueType devra être un type qui implémente l'interface *Comparable* , spécifiquement ceux dont le type générique utilisé dans *Comparable* est le même que la classe de celui qui l'implémente. Il est **fortement recommandé** d'utiliser cette interface pour comparer les objets de type *ValueType* dans votre TP (remplace < , > et ==).

Voici le lien qui amène directement vers sa documentation :
https://docs.oracle.com/javase/7/docs/api/java/lang/Comparable.html

Barème de correction

Partie 1	Réussite des tests	/11.5
	Complexité en temps	/7.5
Qualité du code		/1
		/20

Un chargé s'assurera que votre code ne contourne pas les tests avant de vous attribuer vos points dans la catégorie «Réussite des tests».

Chaque méthode qui ne respecte pas sa complexité temporelle attribuée en commentaire entraînera une pénalité de 1.5 point sur les points de complexité en temps. Un maximum de 7.5 points peut être perdu par ses pénalités.

ERRATUM : ***balance(BinaryNode<ValueType>)***
Cette méthode doit être fait en O(n) et non O(1). Pour les méthodes utilisant *balance*, assumez qu'il est en O(1) pour le calcul de votre complexité temporelle.

Qu'est-ce que du code de qualité ?

- Absence de code dédoublé
- Absence de *warnings* à la compilation
- Absence de code mort
- Respecte les mêmes conventions de codage dans tout le projet
- Variables, fonctions et classes avec des noms qui expliquent leur intention et non leur comportement

Instructions pour la remise

Veuillez envoyer les fichiers .java contenant le code source utile à la résolution des parties 1. Minimalement, le fichier suivant devrait faire partie de votre remise :

- AvlTree.java

Vos fichiers devront être compressés dans une archive *.zip. Le nom de votre fichier compressé devra respecter la formule suivante où MatriculeX < MatriculeY :

inf2010_lab3_MatriculeX_MatriculeY

Chaque jour de retard créera une pénalité additionnelle de 20%. Aucun travail ne sera accepté après 4 jours de retard.