INF2010 - Structures de données et algorithmes

Travail Pratique 3

Arbre binaire de recherche

Département de génie informatique et logiciel

École Polytechnique de Montréal

Automne 2020

Objectifs

- Apprendre le fonctionnement d'un arbre binaire de recherche
- Comprendre la complexité asymptotique d'un arbre binaire de recherche
- Utiliser les concepts associés aux arbres binaires dans un problème complexe

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

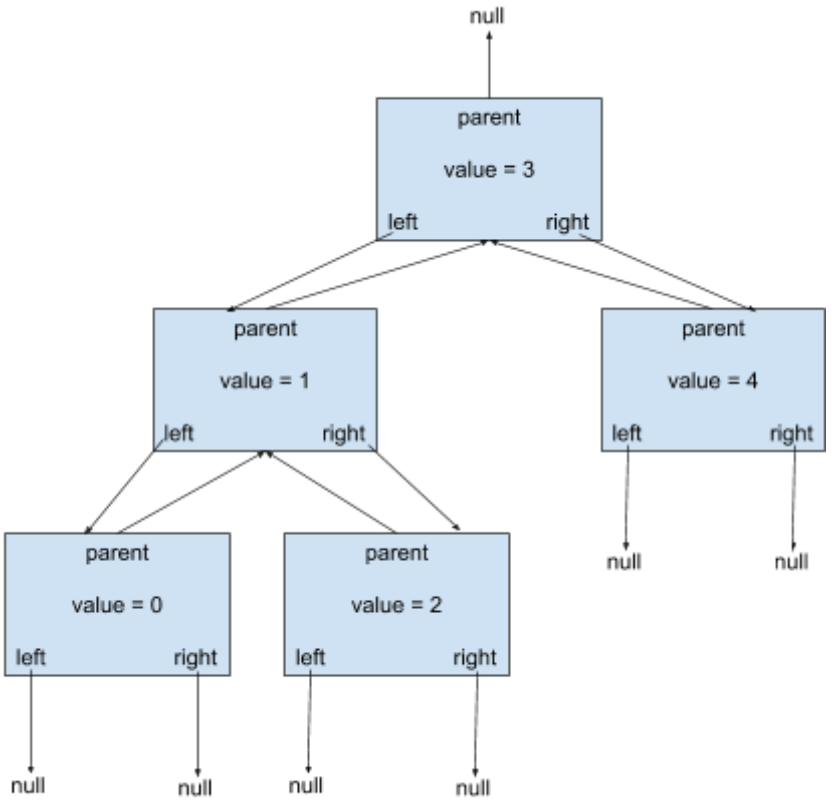
La correction du travail pratique sera partiellement réalisée par les tests unitaires implémentés dans les fichiers sources fournis. La qualité de votre code ainsi que la performance de celui-ci (complexité temporelle) seront toutes deux évaluées par le correcteur. Un barème de correction est fourni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

ATTENTION! ATTENTION! ATTENTION!
Pour ceux qui voudraient déposer leur laboratoire sur **GitHub**, assurez-vous que vos répertoires soient en mode **privé** afin d'éviter la copie et l'utilisation non autorisée de vos travaux. **Un répertoire public peut mener à une sanction de plagiat**.

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 gauche et la hauteur droite soit inférieure 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 ayant un lien ascendant vers leur parent. Voici un exemple d'un arbre AVL utilisant celles-ci.



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

Une note de 0 sera automatiquement attribuée si vous utilisez un arbre binaire d'une librairie quelconque.

Certaines fonctions doivent obligatoirement être implémentées de manière itérative (non-récursif). Celles-ci sont identifiées avec le commentaire suivant `HAS TO BE ITERATIVE, NOT RECURSIVE`

Partie 2 : Problème typique d'entrevue

Entrées

- Matrice de toutes formes ordonnée du plus petit au plus grand de gauche à droite, de haut en bas (attribut *matrix*)

N x N

1	2	3
4	5	6
7	8	9

Forme quelconque

1	3	3		
4	5	6	7	8
9	10	11	12	
13	14			

- Valeur à trouver dans *matrix* (*value*)

Sortie

Valeur booléenne déterminant si l'élément *value* est dans *matrix*

Contraintes

Supposons une matrice (*matrix*) de forme quelconque N x M où N représente le nombre de lignes et M représente le nombre de colonnes pour la ligne la plus longue (ayant le plus de colonnes).

- Complexité temporelle (temps) : O(max(log n, log m))
- Complexité spatiale additionnelle (mémoire) : O(1)
- *matrix* peut contenir des doublons

Pour bien implémenter l'algorithme, suivez les tests contenus dans InterviewTest.java **dans l'ordre de leur définition**.

Il est permis d'utiliser la librairie java.util pour cette partie. Une note de 0 sera attribuée à cette partie si l'étudiant utilise quelconque autre librairie.

Astuces JAVA / IntelliJ

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

Décortiquons les parties intéressantes :

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

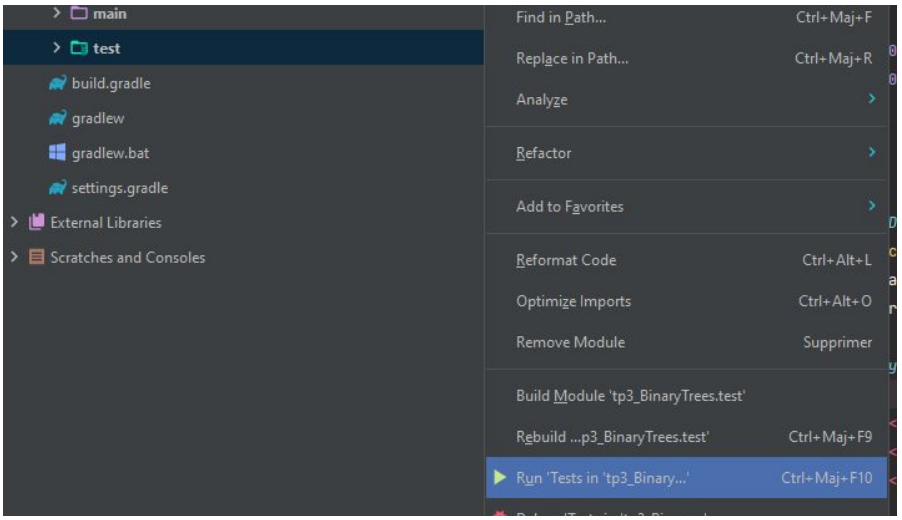
Un type générique permet à une classe de fonctionner avec n'importe quel type. Dans notre situation, on restreint les types qui seront acceptés dans notre classe.

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

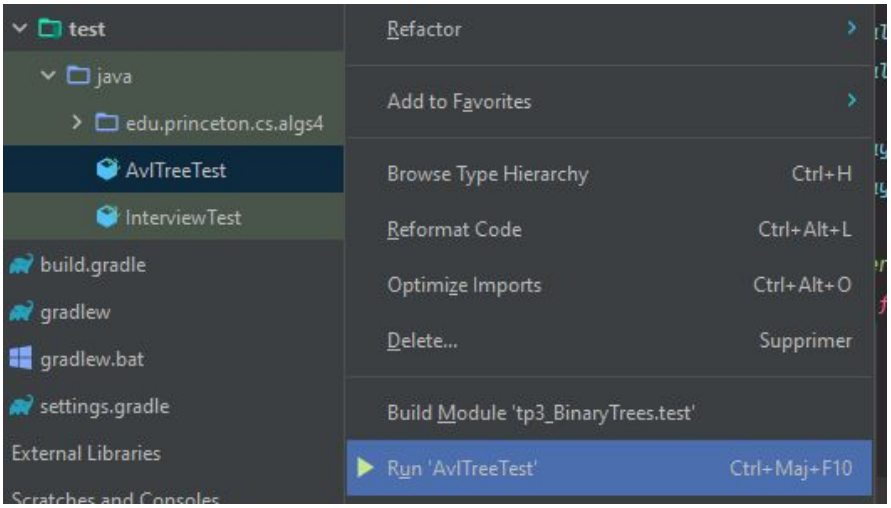
ValueType devra être un type qui implémente l'interface *Comparable* , spécifiquement ceux dont le type générique utilisé dans *Comparable* est le même que la classe de celui qui l'implémente. Il est **fortement recommandé** d'utiliser cette interface pour comparer les objets de type *ValueType* dans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

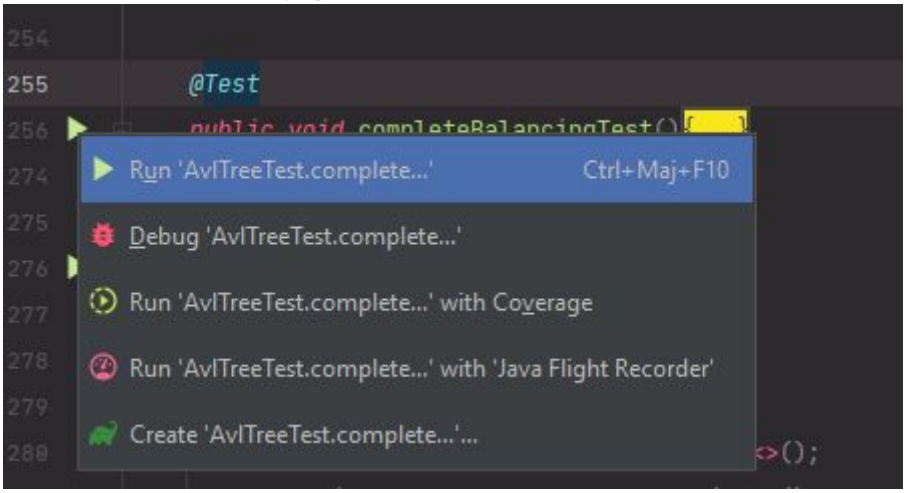
Pour créer une *build configuration* contenant tous les tests



Pour créer une *build configuration* contenant un tester



Pour créer une *build configuration* contenant un test



Barème de correction

Partie 1	Test de complexité	/4
	Autres tests	/7
Partie 2	Complexité	/5
	Autres tests	/3
Qualité du code		/1
		/20

Un chargé s'assurera que votre code ne contourne pas les tests avant de vous attribuer vos points. Le test de complexité n'est pas une garantie que vous aurez tous vos points. La note de la catégorie « Autres tests » est proportionnelle au ratio $\frac{Nombre\ de\ tests\ réussis}{Nombre\ de\ tests}$.

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 deux parties. Minimalement, les fichier AvlTree.java et Interview.java devraient faire partie de votre remise.

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.