** **

Réalisé par :

LE Minh Hao

KEITA Aïssata

TP4 B-ARBRE

RAPPORT STRUCTURE DE DONNEES AVANCEES

Table des matières

[**1.** **Structure/classe de B-arbre** 3](#_Toc28606385)

[**2.** **Expliquez brièvement les choix d’implémentation qui ont été fait :** 4](#_Toc28606386)

[**3. Développez vous-même, ou cherchez sur internet, une structure/classe d’arbre AVL offrant les opérations ci-dessus.** 5](#_Toc28606387)

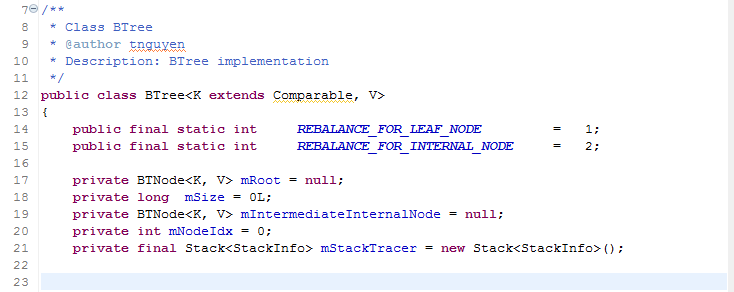
[**3.** **Effectuez des expériences pour comparer l’efficacité en temps et en mémoire des B-arbres et des ABRs.** 6](#_Toc28606388)

1. **Structure/classe de B-arbre**

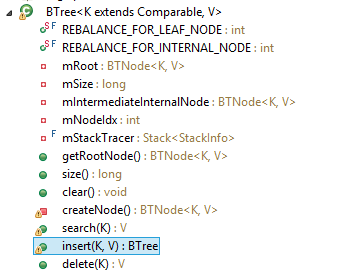
Développez vous-même, ou cherchez sur internet, une structure/classe de B-arbre offrant au moins les opérations suivantes : la création, la recherche d'une clé, l'insertion d'une clé, la suppression d'une clé.

Nous trouve sur Internet, une structure de B-arbre offrant les opérations nécessaires, dans la source - <https://www.codeproject.com/Articles/1158559/B-Tree-Another-Implementation-By-Java>

Ce sont les propriétés de ce B-arbre



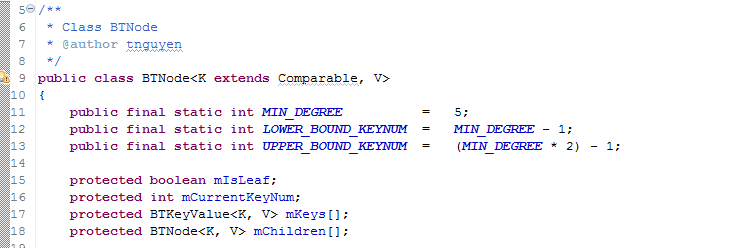
Ce sont les méthodes de ce B-arbre:



Tous les codes sont telecharges sur GitHub dans le lien : <https://github.com/LEMinhHao/paris13_sda>

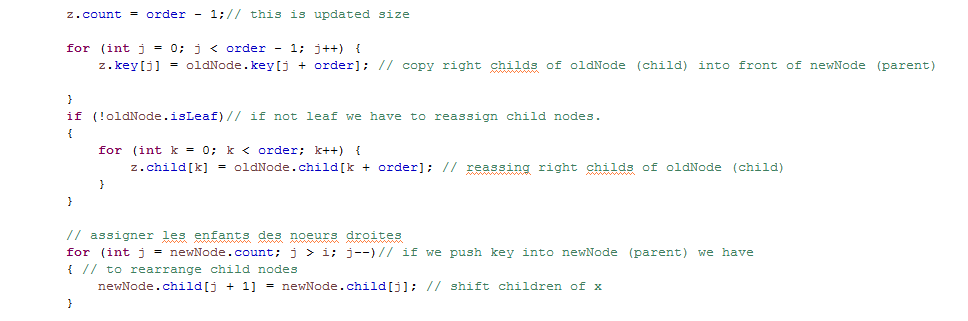
# **Expliquez brièvement les choix d’implémentation qui ont été fait :**

La représentation d’un noeud de l’arbre, la liste de ses clés et celle de ses enfants ; ainsi que l’effet que cela a sur les opérations de fusion et de scindage des nœuds.



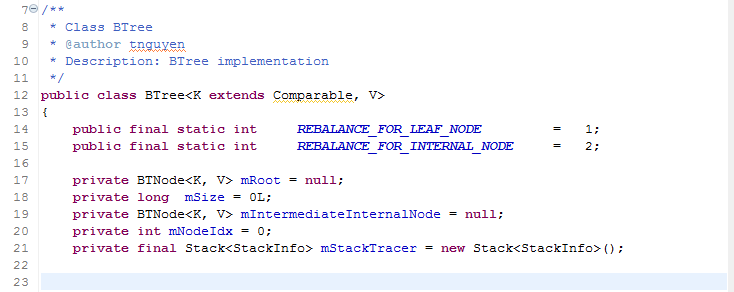
La classe **BTNode** comporte les propriétés comme suivant :

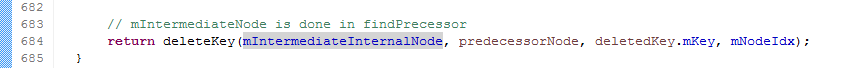
* Un nombre entier *MIN\_DEGREE* stocke la valeur fixe d’ordre de B-tree. Nous va le changer dans la question prochaine.
* Une table fixe **m*Keys[]*** stocke des clés (en nombre entier) de ce nœud. On utilise une table fixe puisque le b-arbre a un nombre fixe de clés (de *t – 1* à *2 \* t – 1*). Ou on peut utiliser une liste chainé pour facilement faire des rotations ou des rétrécissements (dans les opérations de fusion et de scindage des nœuds). Puisque on peut faire plusieurs rotations dans un b-arbre avec la suppression d’une clé.



* Un nombre entier *mCurrentKeyNum* stocke le nombre courant d’élément dans un nœud.
* Une table fixe m*Children[]* stocke des enfants d’un nœud.
* Un booléen m*IsLeaf* vérifie que ce nœud est un feuille ou pas

La classe BTree comporte les propriete comme suivant :

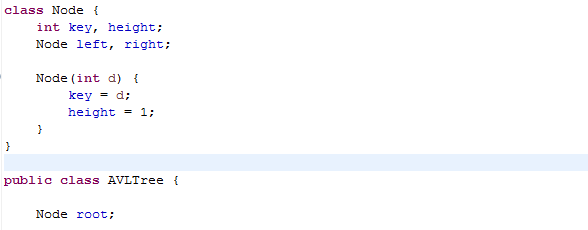


* Les nombres entiers REBALANCE\_FOR\_LEAF\_NODE, REBALANCE\_FOR\_INTERNAL\_NODE défini des types de rééquilibrer.
* La propriete ***mRoot***: BTNode est la racine de B-arbre
* Un nombre entier ***mSize*** est le nombre d’elements de B-arbre
* La propriete ***mIntermediateInternalNode*** stocke un parent-nœud temporaire qui est utilisé pour stocker le parent-nœud et passer globalement dans une suppression récursive.
* 
* La propriété mNodeIdx stocke la position d’une clé dans un nœud en accompagnant la propriete ***mIntermediateInternalNode***dans la méthode de suppression récursive.
* mStackTracer : Stack stocke le chemin d’rééquilibrer et évite une grande opération récursive (qui provoque une erreur StackOverFlow).

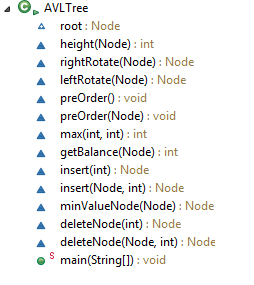
# **3. Développez vous-même, ou cherchez sur internet, une structure/classe d’arbre AVL offrant les opérations ci-dessus.**

Nous utilisons une structure d’arbre AVL offrant les opérations nécessaires:

Ce sont la classe Node et la class AVLTree :



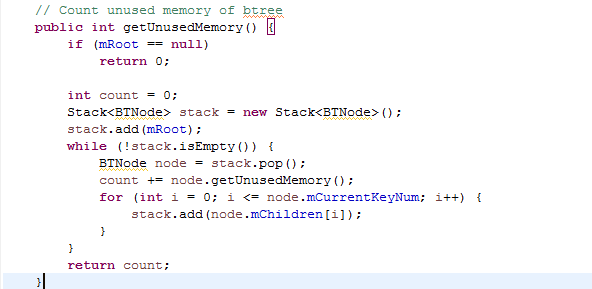
Ce sont ses méthodes nécessaires pour faire les opérations bases :



1. **Effectuez des expériences pour comparer l’efficacité en temps et en mémoire des B-arbres et des ABRs.**

Dans un premier temps, vous ferez uniquement des ajouts dans les deux structures, puis des ajouts et des suppressions. Essayez au moins deux cas : insertion des valeurs croissantes et aléatoires.

On crée une méthode pour compter l’espace mémoire inutilisé de b-arbre. On utilise Stack pour éviter une grande boucle récursive.



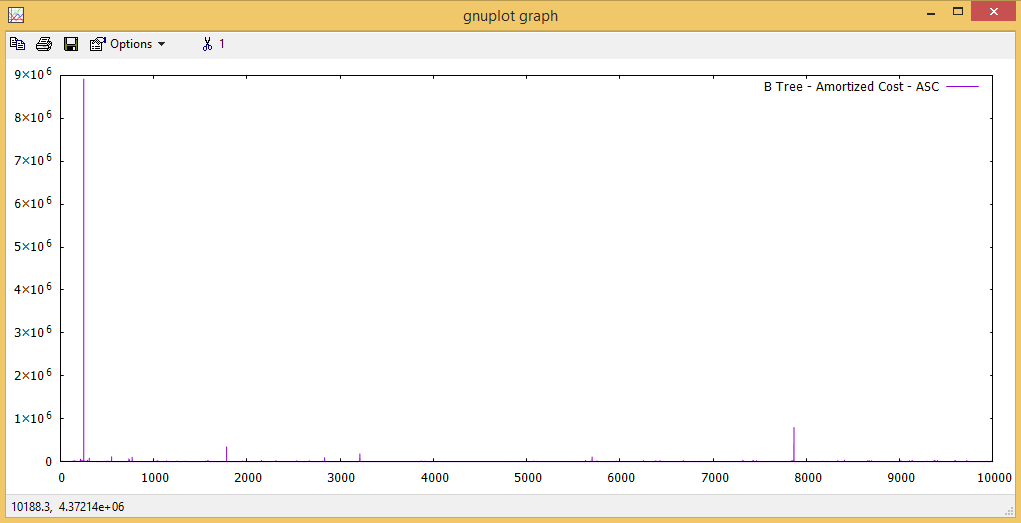
* Dans le cas où on ajoute des cles croissants dans 10 mille operations

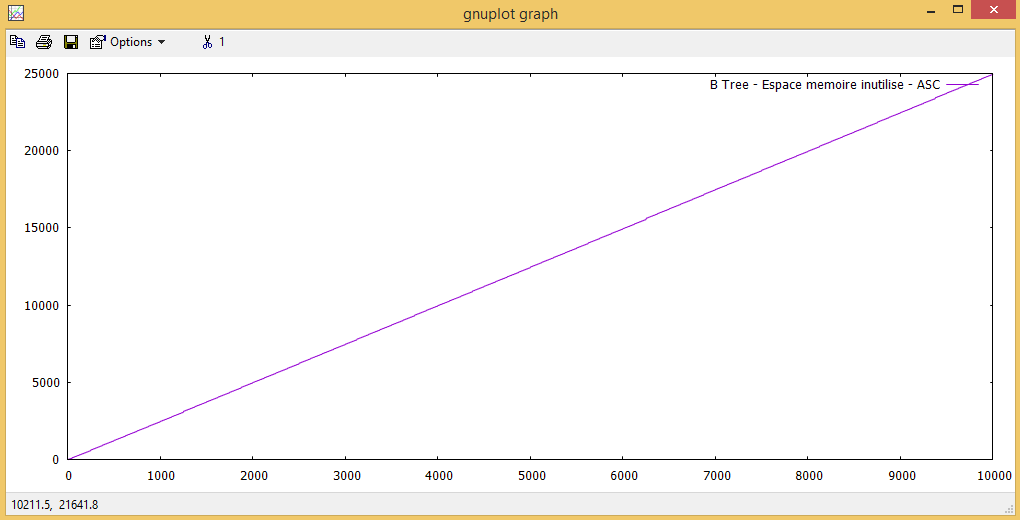
Ce sont les résultats de B-arbre :

* L’espace mémoire inutilisé égale suivant à 2.5 fois le nombre d’élements.

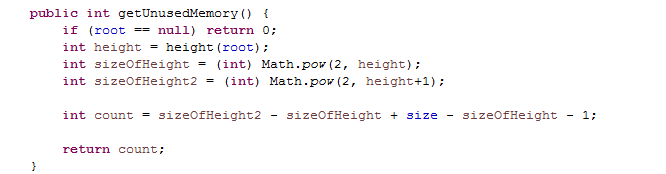
Total cost : 25629800

Average cost : 2562.980

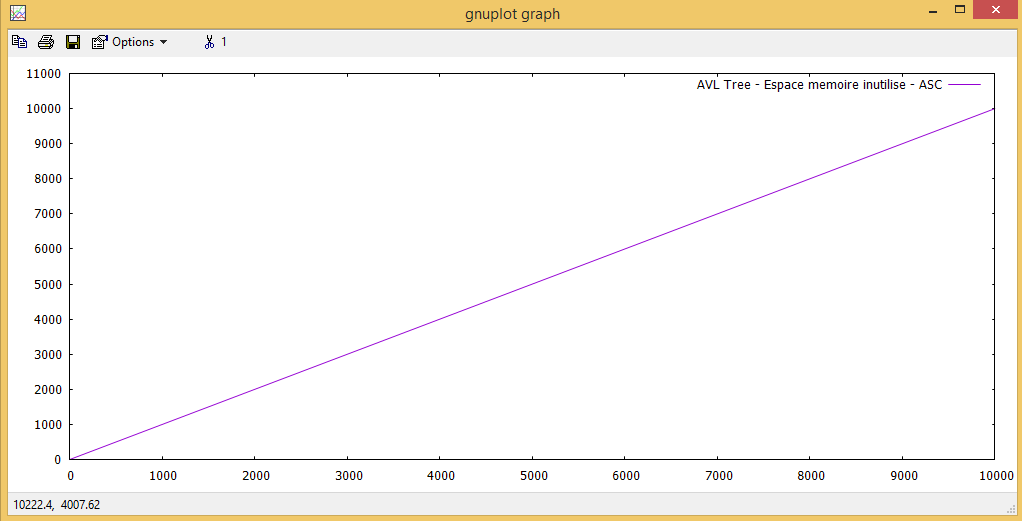


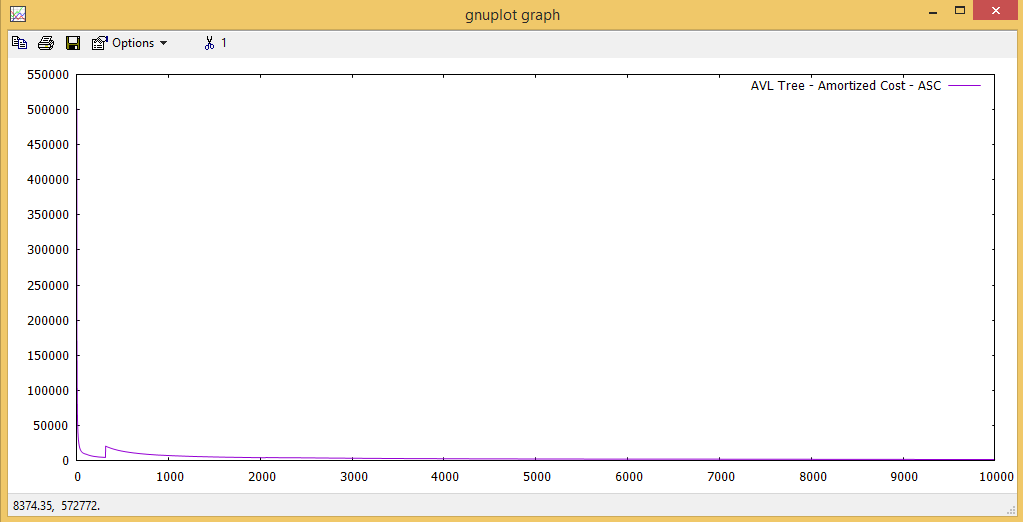


On crée une méthode et ajoute une propriété ***size*** pour compter l’espace mémoire inutilisé d’AVL-arbre. L’idée est à compter les références inutilisées de toutes les feuilles dans AVL-arbre. Parce que les références inutilisées peuvent provoquer l’erreur StackOverFlow.



* L’espace mémoire inutilisé égale au nombre d’éléments + 1 puisque les références inutilisées sont toujours. Par exemple : On a 7 éléments, alors le nombre d’feuilles est 4 et le nombre de références





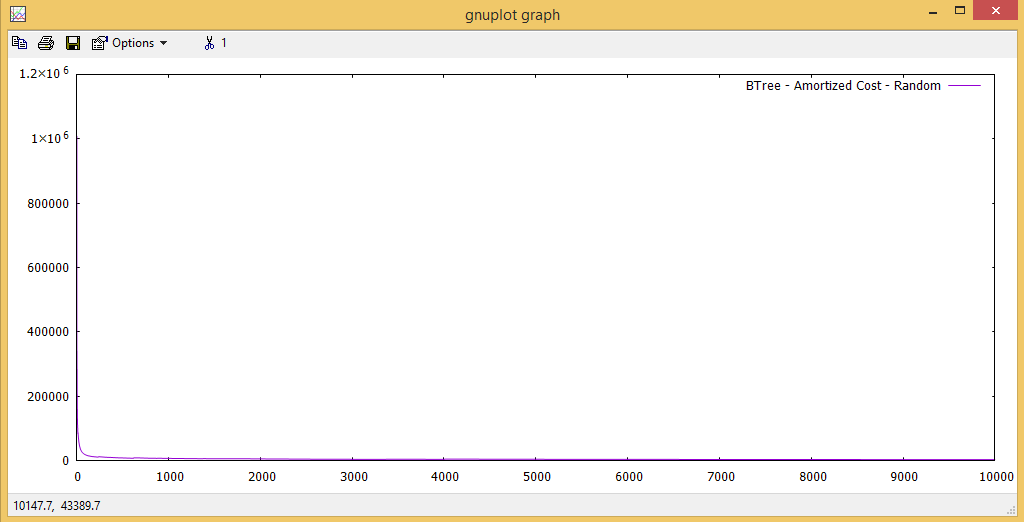
* Dans le cas où on ajoute des clés aléatoires

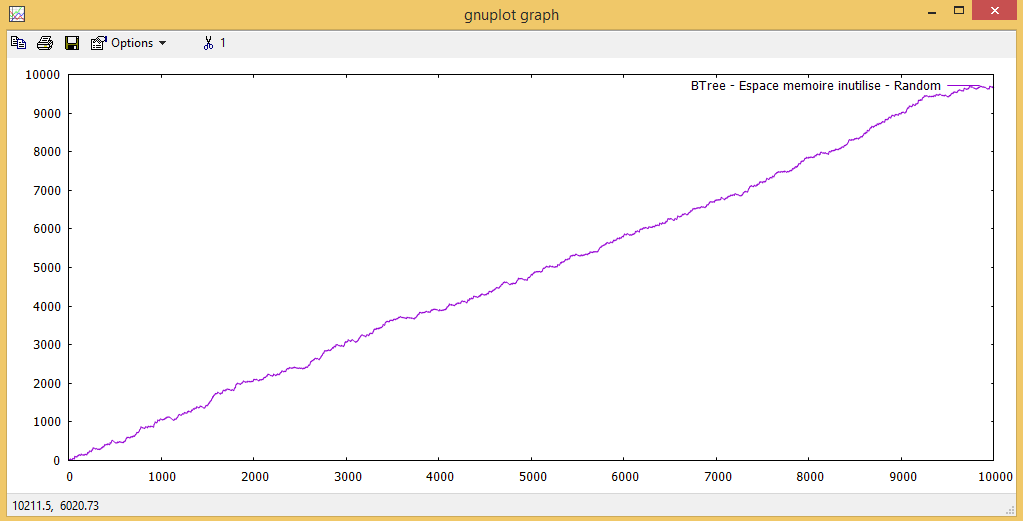
BTree :

* L’espace mémoire inutilisé de ce cas est plus petit que l’espace mémoire inutilisé de premier cas. De plus, l’espace mémoire inutilisé égale semble au le nombre d’éléments de B-arbre

Total cost : 31214732

Average cost : 3121.4732





AVL Tree :

Total cost : 9990869

Average cost : 999.0869

