TP 10 – Arbres binaires de recherche
Année 2019-2020

Un `arbre binaire de recherche` (`ABR`) est un arbre binaire où l'information de chaque noeud est supérieure ou égale à l'information de tous les noeuds de son fils gauche, et inférieure à l'information de tous les noeuds de son fils droit. Pour utiliser des `ABR`, on supposera donc que les informations liées aux noeuds de l'arbre sont comparables.
Ce TP permet de créer des structures de données `ABR` et d'écrire des algorithmes dans le cadre d'applications qui les utilisent.

Exercice 1 – Exercices de base sur les arbres binaires de recherche

Vous vous appuierez sur la modélisation des arbres binaires du TP9. Vous créerez une nouvelle classe ABR (arbre binaire de recherche) qui reprend la classe Arbre avec de nouvelles méthodes relatives aux ABR.

1. Écrivez une fonction `ajoute(self,n)` qui permet d'ajouter un noeud n dans un arbre binaire de recherche.

2. Écrivez une fonction `construit_abr(self,l)` qui renvoie un arbre binaire de recherche construit à partir d'une liste l de valeurs.

3. Créez plusieurs arbres binaires de recherche à partir de listes de valeurs données dans un ordre différent. Que se passe-t-il si vous passez en paramètre une liste triée en ordre croissant ou décroissant ?

4. Écrivez une fonction `inf_abr(self,x)` qui permet de tester si tous les noeuds de l'arbre `self` sont inférieurs ou égaux à x, et une fonction `sup_abr(self,x)` qui permet de tester si tous les noeuds de l'arbre `self` sont strictement supérieurs à x.

5. Écrivez une fonction `test_abr(self)` qui teste si `self` est un `ABR`. Vous ferez attention à tester que pour tout noeud n de l'arbre, tous les noeuds de son sous-arbre gauche (respectivement droit) sont inférieurs ou égaux (resp. strictement supérieurs) à n, et que le sous-arbre gauche (resp. droit) est un `ABR`.

6. Écrivez une fonction `supprime_abr(self, n)` qui supprime une valeur n de l'arbre binaire de recherche `self` si elle existe.

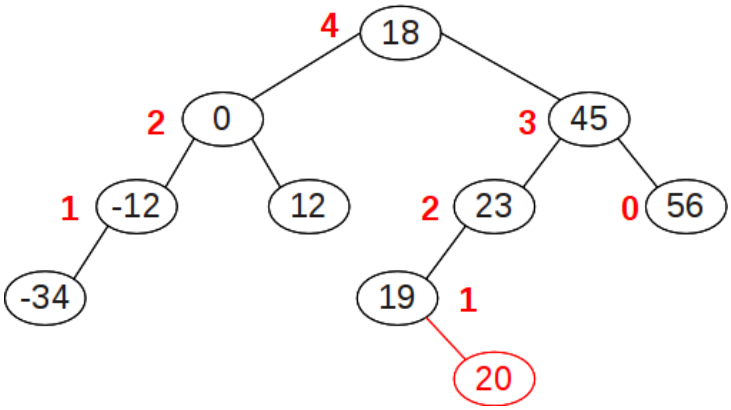
7. Écrivez une fonction `meme_valeurs(self,a)` qui teste si deux arbres binaires de recherche contiennent exactement les mêmes valeurs (mais pas forcément organisées de la même manière).
   Indication : il faut utiliser le fait que les arbres sont des `ABR`.

Exercice 2 – Arbres équilibrés - AVL

Cet exercice est optionnel – Les arbres équilibrés (ou arbres AVL du nom de leurs inventeurs G.M. Adelson-Velskii et E.M. Landis) sont des arbres binaires tels que pour tout noeud de l'arbre, la différence entre les hauteurs du sous-arbre gauche et du sous-arbre droit est au plus égale à 1. On rajoutera par conséquent deux attributs aux noeuds de l'arbre, `hauteur` et `equilibre` définis ainsi :

- l'attribut `hauteur` est défini pour chaque noeud de l'arbre comme étant la taille de la plus grande branche associée à ce noeud (voir arbre ci-dessous) ;
- l'attribut `equilibre` est défini pour chaque noeud de l'arbre comme étant la valeur de la différence en valeur absolue entre la hauteur de son sous-arbre gauche et la hauteur de son sous-arbre droit :
  `equilibre(noeud)` = abs(`hauteur(sous-arbre gauche)` - `hauteur(sous-arbre droit)`).

Un arbre est dit déséquilibré si la valeur de l'attribut `équilibre` est supérieure strictement à 1. Par exemple dans l'arbre donné à la figure suivante, la hauteur de chaque noeud de l'arbre est indiquée en rouge. La valeur d'équilibre du noeud 45 vaut $2 - 0 = 2$, ce qui le caractérise comme un noeud déséquilibré.



Vous ajouterez à la classe Noeud les deux nouveaux attributs, `hauteur` et `equilibre`.

1. Lorsque l'on construit un arbre, ou que l'on modifie cet arbre, par exemple en ajoutant ou en supprimant un noeud, il faut ensuite mettre à jour les attributs `hauteur` et `equilibre`.
   (a) Écrivez la fonction `update_hauteur(self)` qui permet de mettre à jour l'attribut `hauteur` ;
   (b) Écrivez la fonction `update_equilibre(self)` qui permet de mettre à jour l'attribut `equilibre`.
2. Écrivez une fonction `est_AVL(self)` qui renvoie un booléen vrai si l'arbre est équilibré et faux sinon.
3. Testez cette méthode avec des listes d'éléments insérés dans l'arbre dans des ordres différents.
4. Écrivez la fonction `rotate_droit(self)` qui permet de faire une rotation à droite de l'arbre `self`. Le sous-arbre gauche remplace alors la racine courante. De la même façon, écrivez la fonction `rotate_gauche(self)` qui permet de faire une rotation à gauche de l'arbre `self`. Le sous-arbre droit remplace alors la racine courante.