INFO-F103 – Algorithmique 1
Projet 2 : Équilibrage d'arbre binaire

Jérôme De Boeck

Année académique 2020-2021

Enoncé

Un arbre binaire de recherche est dit *équilibré* si pour chaque noeud, la différence entre le nombre de noeud de son sous-arbre gauche et de son sous-arbre droit vaut 0 ou 1. Plus formellement, si n est un noeud d'un arbre T, que T_g^n est son sous-arbre gauche, T_d^n son sous-arbre droit et que $|T|$ est le nombre de noeud de l'arbre T, l'arbre T est équilibré si :

$$|T_g^n| - |T_d^n| \in \{0,1\},\ \forall\, n \in T$$

La recherche d'un élément dans un ABR T se fait en $O(|T|)$ dans le pire cas. Si T est équilibré, la complexité de la recherche devient $O(\log|T|)$, d'où l'intérêt de tenter d'avoir des ABR équilibrés. Pour éviter toute confusion, la notion d'arbre équilibré et d'arbre balancé pour les arbres AVL est légèrement différente. Nous ne considérons ici pas les arbres AVL et leur règle de balancement classique.
La Figure 1 illustre deux ABR. L'exemple de gauche n'est pas équilibré pour deux raisons :

- pour le noeud b, T_g^b contient deux noeuds de plus que T_d^b et $|T_g^b| - |T_d^b| = 2$,
- pour le noeud f, T_g^f contient un noeud de moins que T_d^f et $|T_g^f| - |T_d^f| = -1$.

L'exemple de droite est équilibré car tous les sous-arbres gauches T_g^n contiennent au plus un noeud de plus que le sous-arbre droit correspondant T_d^n.
Si un arbre n'est pas équilibré, il est possible d'effectuer des rotations pour transférer des noeuds du sous-arbre gauche au sous-arbre droit ou inversement tout en préservant l'intégrité de l'ABR. Deux types de rotations sont présentées pour transférer un ou plusieurs noeuds d'un sous-arbre à l'autre. Les détails sont donnés pour le transfère de noeuds du sous-arbre gauche vers le sous-arbre droit.

- La rotation de la racine droite déplace un ou plusieurs noeuds du sous-arbre gauche vers le sous-arbre droit tel qu'illustré sur la Figure 2 en suivant les étapes suivantes :
  - placer b à la racine en conservant son sous-arbre gauche d,
  - placer a et son sous-arbre droit c comme fils droit de la nouvelle racine b,
  - placer e, l'ancien sous-arbre droit de b, comme fils gauche de a.

  Après cette rotation, b est retiré du sous-arbre gauche et a ainsi que le sous-arbre e sont rajoutés au sous-arbre droit. Cette manipulation est rapide mais le nombre de noeuds transférés dépend du nombre de noeuds dans le sous-arbre e.

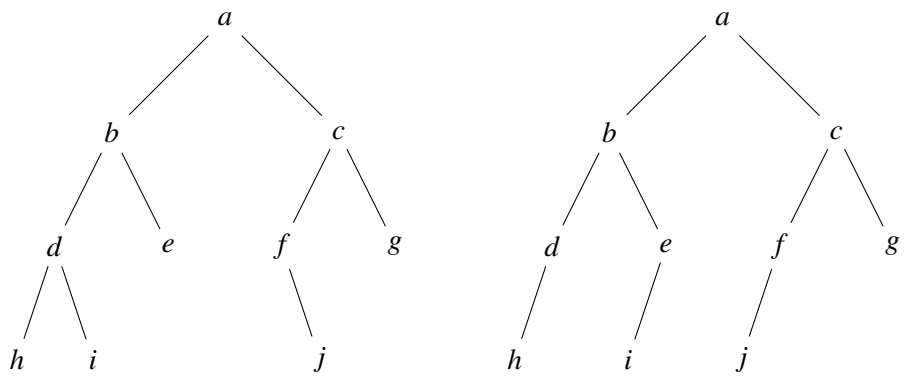

FIGURE 1 – Deux exemples d'ABR, celui de droite étant équilibré.

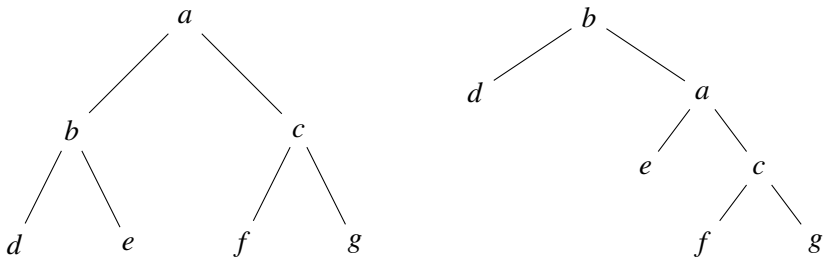

FIGURE 2 – Rotation de la racine droite

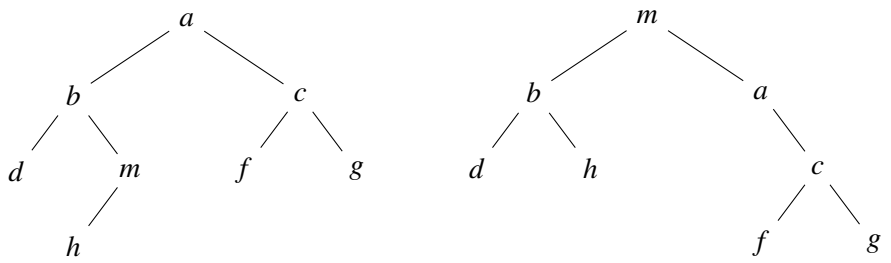

FIGURE 3 – Rotation simple droite

— La rotation simple droite déplace exactement un noeud du sous-arbre gauche vers le sous-arbre droit tel qu'illustré sur la Figure 3 en suivant les étapes suivantes :
  — supprimer le maximum *m* du sous-arbre droit *b* et l'utiliser comme nouvelle racine,
  — placer *a* et son sous-arbre droit *c* comme fils droit de la nouvelle racine *m*,
  — placer *b* (sans la valeur *m*) comme fils gauche de *m*.
  Cette rotation est un peu plus longue en terme de manipulations, ne permet de transférer qu'un seul noeud d'un sous-arbre à l'autre mais le nombre de noeud transféré est fixe.

Pour pouvoir être effectuées, ces rotations requièrent que le racine possède un fils gauche. Les rotations gauche transferrant des noeuds du sous-arbre droit vers le sous-arbre gauche suivent une logique similaire en prenant le minimum du sous-arbre droit pour la rotation simple gauche.

Implémentation

Nous vous demandons de développer un algorithme qui équilibre un ABR en utilisant des rotations de racines et simples (droite et gauche) le plus rapidement possible. Les ABR utilisés sont ceux de la classe `BinaryTree.py` disponible dans le dossier du projet sur l'UV. Une méthode `init_values(self,values)` permet d'initialiser l'arbre en y insérant les valeurs d'une liste `values` avec la méthode `insert`. Vous devez y ajouter les méthodes suivantes :

— `def remove_maximum(self): return maximum`
  Cette méthode supprime le maximum d'un ABR et retourne sa valeur.
— `def rotate_root_right(self) : return None`
  Effectue une rotation de la racine droite sur l'arbre courant ou ne modifie pas l'arbre si celle-ci est impossible.
— `def rotate_root_left(self) : return None`
  Effectue une rotation de la racine gauche sur l'arbre courant ou ne modifie pas l'arbre si celle-ci est impossible.
— `def rotate_simple_right(self) : return None`
  Effectue une rotation simple droite sur l'arbre courant ou ne modifie pas l'arbre si celle-ci est impossible.
— `def rotate_simple_left(self) : return None`
  Effectue une rotation simple gauche sur l'arbre courant ou ne modifie pas l'arbre si celle-ci est impossible.

— `def balance_tree(self) : return None`
Équilibre l'arbre courant. Les manipulations de l'arbre ne peuvent être faites que via les quatre méthodes de rotations ci-dessus durant la procédure sur l'arbre. Elles peuvent donc être appelée dans des fonctions appelées par `balance_tree`. Vous ne pouvez pas supprimer et insérer manuellement des valeurs (sauf pour la suppression nécessaire dans les rotations simple). Tâchez de combiner les rotations de manière à ce que la procédure soit la plus rapide possible.

Vous êtes libre de rajouter d'autres méthodes. Aucun paramètre supplémentaire n'est autorisé. Il est également conseillé d'implémenter des fonctions ou méthodes supplémentaires pour vérifier vos résultats. Il est primordiale que l'ABR reste intact, c'est-à-dire qu'aucune valeur ne disparaisse et que les valeurs dans tout sous-arbre gauche (droit) soient plus petites (grandes) que la racine correspondante. Veillez à obtenir des arbres équilibrés avec la fonction `balance_tree` avant d'envisager des optimisations en terme de rapidité de calcul.

Consignes

Tout votre code doit être contenu dans un seul fichier **BinaryTree.py** que vous compléterez à partir de celui disponible sur l'UV. Veuillez respecter le nom, le format des méthodes et le nom du fichier. Lors de la correction, votre classe `BinaryTree` sera importée via la commande suivante :

```
from BinaryTree import BinaryTree
```

Toutes les méthodes que vous aurez ajoutées seront donc importées. Veillez à ce qu'aucun code ne s'exécute au moment de l'import. Les différentes méthodes seront testée sur un arbre initialisé avec la méthode `init_values`.

Seront évalués la qualité de votre code, la mise en pratique de la matière vue en cours et les optimisations mises en place. Veillez à ce que votre code soit commenté de manière concise pour mettre en valeur ses fonctionnalités et les optimisations appliquées. Dans le cas où l'exécution de votre code en ligne de commande produit une erreur, une note nulle sera reportée pour la partie exécution. Veillez donc à tester toutes vos méthodes à partir d'un autre fichier en y important votre classe avant la remise.

Pour remettre votre projet sur l'UV, nous vous demandons de

— créer localement sur votre machine un répertoire de la forme NOM_Prenom (exemple :DUPONT_Jean) dans lequel vous mettez le fichier `BinaryTree.py` à soumettre
— compresser ce répertoire via un utilitaire d'archivage produisant un .zip (aucun autre format de compression n'est accepté)
— de soumettre le fichier archive .zip, et uniquement ce fichier, sur l'UV

Le projet est à remettre pour le 2 avril 2021 à 23h sur l'UV. Pour toute question contactez-moi à l'adresse jdeboeck@ulb.ac.be. Tout manquement aux consignes ou retard sera sanctionné d'un **0/10**.