Arbre Binaire de Recherche (ABR)

Un arbre binaire est ici défini comme suit :

- un arbre vide correspond à la valeur `None`
- La racine d'un arbre non vide est une liste constituée de 3 éléments
  - la valeur
  - la racine du sous-arbre gauche
  - la racine du sous-arbre droit

Un arbre binaire de recherche respecte de plus la contrainte suivante:

- pour tout nœud de l'arbre,
  - les valeurs situées dans le sous-arbre gauche sont plus petites que la valeur du nœud
  - les valeurs situées dans le sous-arbre droit sont plus grandes que la valeur du nœud

Nous implémentons dans ce TP un arbre binaire de recherche.

Ouvrez l'éditeur Pycharm et créez un nouveau projet.

Dans ce projet, créez un fichier python `arbreBinaireRecherche.py` et un fichier de test `testeABR.py`

1. Pour commencer

L'accès aux valeurs de l'arbre se fait à partir la racine. Pour commencer, nous créons 3 étiquettes:

```
INDEX = 0
GAUCHE = 1
DROIT = 2
```

- Créez une fonction `init_arbre` qui crée un arbre de recherche vide.
- Testez cette fonction dans le fichier de test (vérifiez que l'arbre créé est bien vide)
- Créez une fonction `feuille` qui prend en argument une valeur et retourne une feuille
- Testez cette fonction dans le fichier de test (vérifiez que la valeur contenue dans la feuille est correcte, et que les fils gauche et droit sont nuls)

2. Insérer des valeurs

L'insertion de valeurs se fait généralement de manière récursive, comme suit:

```
algo : insérer_rec
données :
  * racine
  * val
début:
    si racine est nul:
        retourner feuille(val)
    sinon:
```

```
        si val < racine[INDEX]:
            racine[GAUCHE] <-- insérer_rec(racine[GAUCHE], val)
        sinon:
            racine[DROIT] <-- insérer_rec(racine[DROIT], val)
        retourner racine
```

- Ajoutez la fonction `insérer_rec` dans `arbreRecherche.py`
- Testez-la à l'aide du fichier de test, en particulier:
    - vérifier que l'insertion d'une valeur dans un arbre vide produit un arbre non vide
    - vérifiez qu'après l'insertion de plusieurs valeurs, la valeur de la racine est correcte
    - affichez l'arbre obtenu après l'insertion de plusieurs valeurs (sous forme de liste)

3. Affichage en ordre

Écrivez et testez une fonction qui affiche les valeurs de l'arbre selon le parcours "en ordre".

4. Recherche

- Écrivez une fonction de recherche récursive `recherche_rec` qui prend en argument une valeur et retourne `True` si la valeur est présente dans l'arbre et `False` si elle est absente.
- Testez cette fonction dans le fichier de tests

5. Suppression

- Ecrivez et testez une fonction `cherche_max` qui retourne le plus grand élément d'un arbre
- Aidez-vous de la fonction `cherche_max` pour écrire une fonction de suppression récursive `supprime_rec` qui prend en argument une valeur et supprime la valeur si celle-ci est présente dans l'arbre et ne change rien sinon.
- Testez cette fonction dans le fichier de tests

PS : si vous avez le temps, vous pouvez également tester la suppression itérative vue en TD.

6. Affichage en arbre

Pour bien comprendre l'effet de la suppression, essayez de produire un affichage un peu plus joli pour l'arbre, par exemple :

```
   |   |   |   |41
   |   |   |32-|
   |   |27-|
   |   |   |24
   |23-|
   |   |   |16
   |   |14-|
```

```
   |   |   |12
10-|
   |   |   |8
   |   |7--|
   |   |   |6--|
   |   |   |   |5
   |3--|
   |   |2--|
   |   |   |1
```

ancien sujet

From:
https://wiki.centrale-med.fr/informatique/ - **WiKi informatique**

Permanent link:
https://wiki.centrale-med.fr/informatique/tc_info:tp7

Last update: **2019/09/28 16:17**