TD-TP "Arbres Binaires de Recherche"

Ce "TD-TP" est, comme son nom l'indique, constitué une partie TD à faire sur papier (il est conseillé de faire aussi l'exercice 7 sur machine) & d'une partie TP à faire sur machine.

Il n'est pas nécessaire de faire tout le TD avant d'attaquer le TP, néanmoins, avant de programmer un algorithme, il faut l'écrire à la main.

Partie TD

Un **Arbre Binaire** est un arbre tel que chaque sommet a au plus deux fils. Les arbres binaires sont très utilisés car ils permettent de modéliser un grand nombre de situations ; de plus, les arbres non binaires sont plus complexes à implémenter (*i.e.* programmer effectivement).

1. Caractéristiques d'un arbre binaire

Donnez des algorithmes (avec leur complexité) qui calculent:

- Le nombre de sommets d'un arbre,
- Le nombre de feuilles d'un arbre,
- La hauteur d'un arbre.

Indication.
On pourra se restreindre aux arbres binaires

Un **Arbre (Binaire) de Recherche** est un arbre binaire dont les sommets sont indexés par un ensemble ordonné (*e.g.* des nombres) & tel que, pour chaque sommet s, les descendants gauches de s aient une étiquette plus petite que celle de s & les descendants droits une étiquette plus grande.

2. Dessin

Dessinez un arbre de recherche.
Indication.

3. Parcours

Que donnent les trois parcours classiques (En-ordre, Pré-Ordre, Post-Ordre) sur un arbre de recherche ?

Indication

4. Fonctions de base

Donnez des algorithmes (avec leurs complexités) qui :

- Détermine si une valeur x est dans un arbre de recherche
- Insère une valeur x dans un arbre de recherche. Indication
- Supprime une valeur x d'un arbre de recherche. Indication

5. Création

Donnez un algorithme qui, à partir d'une liste de nombres, crée un arbre de recherche (Indication). Quel est cet algorithme (Indication)?

Pour rechercher une valeur dans un tableau *trié*, on utilise généralement la **recherche dichotomique**, qui consiste à tester la valeur "du milieu", éliminer une des deux moitié du tableau & recommencer sur l'autre moitié jusqu'à ce que le tableau soit de taille 1.

6. Recherche dichotomique 1

Quel est le lien entre recherche dichotomique & arbres de recherche ?

7. Recherche dichotomique 2

Écrivez la recherche dichotomique (de préférence de manière itérative). Attention, la recherche dichotomique est un algorithme très simple mais sur lequel on fait beaucoup d'erreurs — Testez bien votre programme.

8. Choix 1

Il existe deux versions de la recherche dichotomique : soit on arrête l'algorithme dès qu'on a trouvé la valeur, soit on va systématiquement jusqu'à avoir un (sous-)tableau de taille 1 & on vérifie s'il contient la valeur recherchée. Quelle est la version la plus efficace ? Indication

9. Choix 2

Avantages & inconvénients des arbres de recherche par rapport aux tableaux triés ? Réponse

Partie TP

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

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
```