TP nº 8 - Arbres binaires

Exercice 1. *Fonctions récursives*

1. Recopiez les classes suivantes correspondant plus ou moins à celles qui ont été vues en cours :

```
class Noeud:
        pass
def noeud(x):
        n = Noeud()
        n.valeur = x
        n.gauche = None
        n.droit = None
        return n
class Arbre:
        pass
def arbre(n=None):
        a = Arbre()
        a.racine = n
        return a
```

2. Écrivez les fonctions élémentaires de manipulations d'arbres suivantes :
- `vide(a)` qui renvoie `True` si l'arbre `a` est vide et `False` sinon (attention, d'après la définition donnée un arbre a toujours un nœud racine mais ce nœud est éventuellement vide, c'est ça qu'il faut regarder) ;
- `gauche(a)` et `droit(a)` qui renvoient respectivement le sous-arbre gauche et le sous-arbre droit d'un arbre binaire `a` (attention, il faut renvoyer un arbre donc trouver le fils gauche ou droit de la racine de `a` puis renvoyer un arbre ayant ce nœud pour racine) ;
- `racine(a)` qui renvoie la valeur de la racine de l'arbre ;
- `cons(x, a1, a2)` qui renvoie un arbre dont la racine contient la valeur `x` et dont le fils gauche est l'arbre `a1` et le fils droit est l'arbre `a2`.

Remarque : toute ces fonctions sont très courtes.

3. Dessinez l'arbre `a` construit par les instructions suivantes :

```
>> g = cons(1, cons(0, arbre(), arbre()), arbre())
>> d = cons(4, cons(3, arbre(), arbre()), cons(5, arbre(), arbre()))
>> a = cons(2, g, d)
```

4. Écrivez les fonctions suivantes de manière récursive en n'utilisant que les primitives de la question 2 et la fonction `arbre(n)` :
- `profondeur(a)` qui renvoie la profondeur de l'arbre (la longueur du plus long chemin descendant de la racine vers une feuille) ;
- `nb_noeuds(a)` qui renvoie le nombre de nœuds de l'arbre `a` ;
- `somme(a)` qui renvoie la somme de toutes les valeurs contenues dans l'arbre ;
- `incremente(a)` qui renvoie l'arbre `a` dans lequel toutes les valeurs ont été augmentées de 1 ;
- `hierarchie(a)` qui vérifie que pour tout nœud de l'arbre, la valeur du nœud est supérieure ou égale à la valeur de chacun de ses fils non vides. La fonction renvoie `True` si cette propriété est vraie et `False` sinon.

Exercice 2. *Parcours*

Nous allons nous intéresser ici au différents moyens de parcourir un arbre, c'est-à-dire de passer sur chacun des sommets. Nous allons donc écrire des fonctions permettant d'afficher toutes les valeurs contenues dans un arbre en faisant attention à l'ordre dans lequel les nœuds sont parcourus.

1. Écrivez une fonction qui affiche toutes les valeurs contenues dans un arbre (sans vous préoccuper de l'ordre dans lequel elles sont affichées) puis décrivez sur un exemple l'ordre dans lequel l'arbre est parcouru.

2. Écrivez une fonction qui parcourt un arbre en vérifiant la condition suivante : pour chaque nœud de l'arbre, le nœud est visité avant ses fils (dans notre exemple, sa valeur est affichée avant celle de ses fils) et l'ensemble des nœuds de son sous-arbre gauche doivent être visités avant son fils droit. Remarque : il n'existe qu'un seul ordre de parcours de l'arbre vérifiant cette propriété.

3. Écrivez une fonction qui parcourt un arbre en vérifiant la condition suivante : un nœud est toujours visité après tous ses descendants et les descendants gauche d'un nœud sont visités avant ses descendants droits (il n'y a ici encore qu'un seul ordre possible).

4. Testez vos fonctions sur l'exemple de la question 3.

Exercice 3. *Fonctions récursives avancées*

Écrivez les fonctions récursives suivantes (toujours en utilisant les primitives de manipulation d'arbres) :
- `applique(f, a)` qui applique la fonction f à chacune des valeurs de l'arbre ;
- `applique_prof(f, a)` qui applique la fonction f à toutes les valeurs de l'arbre, autant de fois que la profondeur du nœud considéré (on applique une fois la fonction à la racine, deux fois à chaque fils de la racine, trois fois aux fils des fils de la racine, etc.) ;
- `grand_chemin(a)` qui renvoie la plus grande somme possible de valeurs sur un chemin descendant dans l'arbre de la racine à une feuille ;

Exercice 4. *Arbres binaires de recherche*

En manipulant directement les arbres (on peut accéder directement aux nœuds et à leurs champs `valeur`, `gauche`, `droit` et `parent`) écrivez les fonctions suivantes sur les arbres binaires de recherche (vous pouvez éventuellement écrire et utiliser des fonctions annexes pour simplifier l'écriture des fonctions demandées) :
- `memes_valeurs(a1, a2)` qui teste si deux arbres binaires de recherche contiennent exactement les mêmes valeurs (mais pas nécessairement organisées de la même manière). Indication : il faut bien exploiter le fait que les arbres sont des arbres de recherche ;
- `sous_arbre(a1, a2)` qui teste si l'arbre `a2` est un sous-arbre de l'arbre `a2`, c'est-à-dire s'il existe un nœud n de `a1` tel que le sous-arbre enraciné en n soit égal à `a2`.