M1 MSI
INITIATION ALGORITHMIQUE ET PROGRAMMATION
LANGAGE PYTHON
TRAVAUX PRATIQUES
Brice Mayag

TP2 : ARBRES BINAIRES ET DICTIONNAIRES

Notion d'arbre binaire

L' arbre est un des concepts algorithmiques les plus importants de l'informatique. C'est une notion très courante dans la vie quotidienne : arbres généalogiques, organigrammes, ... C'est également une structure récursive : les suivants sont aussi des arbres.

Un arbre est une collection (finie), et éventuellement vide, de noeuds de même type et d'arêtes.
— Noeuds : données contenues dans l'arbre
— Arêtes : lien entre deux noeuds.

Un arbre est défini par :
— Une Racine : noeud distingué de l'arbre (arbre enraciné)
— 0,1,2 ou plusieurs sous-arbres disjoints : descendants de la racine.

Quelques définitions :
— Feuilles : noeuds sans descendants
— Fils d'un noeud : racine d'un descendant de ce noeud.
— Noeud interne : noeud avec descendance.
— Branche (ou chemin) d'un arbre : suite de noeuds distincts dont les noeuds successifs sont reliés par une arête.
— Longueur d'une branche : nombre de ses arêtes

Un arbre binaire sur un ensemble fini est soit vide, soit l'union disjointe de noeud appelé sa racine, d'une arbre binaire appelé sous-arbre gauche, et d'un arbre binaire appellé sous-arbre droit.

Exercice

Nous donnons ci-dessous l'exemple de l'arbre binaire `a1` ayant l'entier 4 comme racine, 10 nœuds dont 4 feuilles et 5 nœuds internes. Notons que les arbres vides sont représentés comme des dictionnaires vides, et qu'un arbre non vide contient exactement les 3 clés suivantes : 'rac' pour racine, 'g' pour désigner le sous-arbre gauche, et 'd' pour le sous-arbre droit.

```
a1 = { 'rac' : 4,
       'g' :{'rac' : 2,
             'g' : {},
             'd' : { 'rac' : 3, 'g' : {}, 'd' : {}}},
       'd' : {'rac' : 7,
              'g' : {'rac' : 5, 'g' : {}, 'd' : {'rac' : 6, 'g' : {}, 'd' : {}}},
              'd' : {'rac' : 10,
                     'g' : {'rac' : 9, 'g' : {}, 'd' : {}},
                     'd' : {'rac' : 14, 'g' : {}, 'd' : {}}}}}
```

1. Écrire un fonction `est_abr(a)` qui teste si `a` est un objet Python représentant un arbre binaire, conformément à la représentation choisie ci-dessus.

2. Écrire une fonction `build_abr(r,g,d)` qui à partir d'une valeur `r`, un arbre `g`, et un arbre `d`, construit un arbre de racine `r`, de sous-arbre gauche `g` et de sous-arbre droit `d`.
3. Écrire les fonctions d'observation des arbres `est_vide`, `racine`, `gauche`, `droit` qui respectivement teste si un arbre est vide, fournit la valeur de la racine d'un arbre non vide, fournit le sous-arbre gauche d'un arbre non vide, et enfin le sous-arbre droit.
4. Écrire les fonctions usuelles portant sur les arbres :
   (a) `hauteur` (longueur maximale du chemin de la racine à une feuille)
   (b) `est_feuille` prédicat testant si un arbre est réduit à une feuille, i.e. une racine avec des sous-arbres vides
   (c) `nombre_feuilles` comptant le nombre de feuilles d'un arbre
   (d) `nb_noeuds_internes` comptant le nombre de nœuds internes, i.e. les nœuds ayant au moins un sous-arbre non vide
5. Écrire les fonctions de parcours des arbres :
   (a) parcours infixe (le sous-arbre gauche est parcouru avant la racine, puis le sous-arbre droit. Avec l'arbre `a1`, cela donne : [2, 3, 4, 5, 6, 7, 9, 10, 14])
   (b) parcours préfixe (la racine est parcourue d'abord, puis le sous-arbre gauche et enfin le sous-arbre droit. Avec l'arbre `a1`, cela donne : [4, 2, 3, 7, 5, 6, 10, 9, 14])
   (c) parcours suffixe (les sous-arbres gauche et droit sont parcourus d'abord, puis la racine. Avec l'arbre `a1`, cela donne : [3, 2, 6, 5, 9, 14, 10, 7, 4])