EA4 – Éléments d'algorithmique
TP n° 7 : arbres binaires

Dans ce TP, on représente un arbre binaire par une liste de nœuds, chaque nœud étant une liste de taille 4 au format

`[étiquette, filsGauche, filsDroit, père]`,

où les trois derniers champs sont, s'ils existent, les indices des nœuds correspondants dans la liste, et `None` sinon. Chaque nœud de l'arbre sera désigné par son indice dans la liste.

Le fichier `tp7.py` définit les fonctions suivantes :

- `arbreBinaireDeFichier(fichier)` qui construit la liste correspondant à l'arbre décrit par le fichier dont le nom est passé en paramètre ;
- `dessineArbreBinaire(arbre)` qui crée un fichier `/tmp/arbre.pdf` représentant l'arbre passé en paramètre ;
- `completeArbreBinaire(arbre)` qui ajoute des feuilles tout autour de l'arbre, pour obtenir un arbre binaire complet ; chaque feuille vaut `[None, None, None, i]`, où `i` est l'indice de son père.

Vous trouverez également sur Moodle un exemple de fichier décrivant un arbre binaire de recherche, `planetes.txt`.

Vous constaterez que l'affichage des arbres binaires non complets n'est pas satisfaisant, car l'utilitaire employé ne fait pas de distinction entre un fils unique gauche et un fils unique droit. Pour obtenir un dessin respectant la latéralité, il faut donc compléter l'arbre avant de le dessiner.

L'intégralité de ce TP devra être rendue sur Moodle avant dimanche 16h (heure de Paris), sous forme d'une archive `tp7.tar` ou `tp7.zip`.

Conventions : L'arbre vide (non complet) est représenté par la liste vide `[]`, et l'arbre vide complété par `[[None, None, None, None]]`. La racine d'un arbre est le seul node dont le père est `None`.

Exercice 1 : accesseurs

1. Écrire une fonction `estVide(arbre)` qui renvoie `True` si l'`arbre` est vide, et `False` sinon.
2. Écrire des fonctions `etiquette(arbre, i)`, `filsGauche(arbre, i)`, `filsDroit(arbre, i)`, `pere(arbre, i)` retournant respectivement l'étiquette et les indices du fils gauche, du fils droit et du père du nœud d'indice `i` dans l'`arbre`.
3. Écrire une fonction `estRacine(arbre, i)` qui retourne `True` si le nœud d'indice `i` est la racine de l'`arbre`, et `False` sinon.
4. Écrire des fonctions `estFilsGauche(arbre, i)` et `estFilsDroit(arbre, i)` qui retourne `True` si le nœud d'indice `i` est le fils gauche (ou droit) de son père, et `False` sinon.
5. Écrire une fonction `estFeuille(arbre, i)` qui retourne `True` si le nœud d'indice `i` est une feuille de l'`arbre`, et `False` sinon – en particulier, si l'`arbre` est un arbre binaire complété, elle doit répondre `True` uniquement pour les feuilles ajoutées (dont l'étiquette est `None`).

Toutes ces fonctions doivent être testées, par exemple sur les arbres définis dans `tp7.py`.

Exercice 2 : parcours

1. Écrire une fonction `profondeur(arbre, i)` qui calcule la profondeur du nœud d'indice `i` dans l'`arbre`.

Dans les questions suivantes, si l'`arbre` a été complété, on ne tiendra pas compte des feuilles ajoutées.

2. Écrire une fonction `hauteur(arbre)` qui calcule la hauteur de l'`arbre` avec une meilleure complexité que la fonction `hauteurNaive(arbre)` fournie. Pour cela, vous pouvez écrire une fonction auxiliaire récursive `hauteurAux(arbre, i)` calculant la hauteur du sous-arbre de racine `i`.
3. Écrire une fonction ***non récursive*** `parcoursPrefixe(arbre)` qui calcule la liste des étiquettes des nœuds de l'`arbre` en ordre préfixe.
4. Écrire une fonction `estUnABR(arbre)` qui renvoie `True` si les étiquettes des nœuds de l'`arbre` vérifient les conditions d'un ABR pour l'ordre lexicographique.