Arbres généraux (General Trees)

1 implementation

La "classique" : par n-uplets

La plus simple : la liste des sous-arbres est une simple liste Python (liste contigüe). L'arbre sera représenté par un objet `T: Tree` représentant le nœud racine contenant les "champs" suivants :

— une clé `key`,

— une liste d'arbres (n-uplet) `children (list)`.

`nbchildren` n'est pas nécessaire, mais permet de simplifier l'écriture / la lecture des fonctions (plus proche des algos du cours).

Avantage de cette implémentation ? L'accès direct aux fils, qui pourra être utile s'ils sont "triés" (*arbres de recherche* qui seront vus au prochain td. . .).

algopy/tree.py

```python
class Tree:
    def __init__(self, key=None,
                 children=None):
        self.key = key
        if children == None:
            self.children = []
        else:
            self.children = children
    @property
    def nbchildren(self):
        return len(self.children)
```

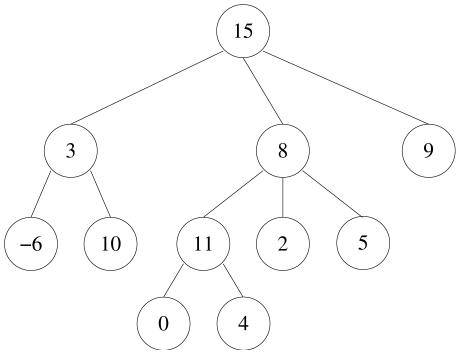

FIGURE 1 – Arbre général T_1

2 Mesures

Exercice 1 (Taille (Size))

1. Donner la définition de la taille d'un arbre.
2. Écrire une fonction qui calcule la taille d'un arbre.

Exercice 2 (Hauteur (Height))

1. Donner la définition de la hauteur d'un arbre.
2. Écrire une fonction qui calcule la hauteur d'un général.

3 Parcours

Exercice 3 (DFS : Parcours en profondeur (Depth First Search))

1. Quel est le principe du parcours en profondeur d'un arbre général ?
2. Donner les listes des éléments rencontrés dans les ordres préfixe et suffixe lors du parcours profondeur de l'arbre de la figure 1. Quels autres traitements peut-on faire ?
3. Écrire le parcours en profondeur (insérer les traitements).

Exercice 4 (BFS : Parcours en largeur (Breadth First Search))

1. Quel est le principe du parcours en largeur d'un arbre ?
2. Comment repérer les changements de niveaux lors du parcours en largeur ?
3. Écrire une fonction qui affiche les clés d'un arbre général niveaux par niveaux, un niveau par ligne.

4 Applications

Exercice 5 (Représentation par *listes*)

Soit un arbre général A défini par $A =< o, A_1, A_2, ..., A_N >$. Nous appellerons *liste* la représentation linéaire suivante de A : (o A_1 A_2 ... A_N).

1. (a) Donner la *représentation linéaire* de l'arbre de la figure 1.
   (b) Soit la *liste* `(12(2(25)(6)(-7))(0(18(1)(8))(9))(4(3)(11)))`, dessiner l'arbre général correspondant.
2. Écrire la fonction qui construit à partir d'un arbre sa *représentation linéaire* (sous forme de chaîne de caractères).

Exercice 6 (Profondeur moyenne externe (External Average Depth))

1. Donner la définition de la profondeur moyenne externe d'un arbre.
2. Écrire une fonction qui calcule la profondeur moyenne externe d'un arbre.

5 Bonus

Exercice 7 (Famille nombreuse – *Contrôle S3 - 2021*)

Écrire la fonction `morechildren(`T`)` qui vérifie si chaque nœud interne de l'arbre T a strictement plus de fils que son père, avec l'implémentation *premier fils - frère droit*.

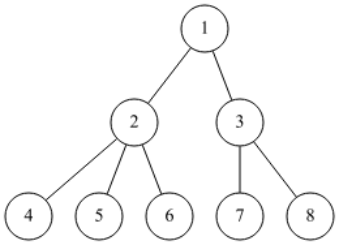

FIGURE 2 – Arbre `T1`

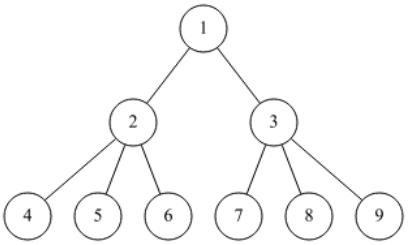

FIGURE 3 – Arbre `T2`

Exemples d'applications avec les arbres des figures 2 (`T1`) et 3 (`T2`) :

```
>>> morechildren(T1)
False
>>> morechildren(T2)
True
```

Exercice 8 (Chargement d'arbres depuis des fichiers)

Pour stocker les arbres dans des fichiers textes (`.tree`) nous utilisons la représentation par *listes* vue à l'exercice 5 ($A =< o, A_1, A_2, ..., A_N >$ est représenté par (o A_1 A_2 ... A_N).)

Écrire la fonction qui construit l'arbre à partir de la *liste* (type `str`) dans les deux implémentations, voir `algopy/tree.py` et `algopy/treeasbin.py`.