Arbres généraux (General Trees)

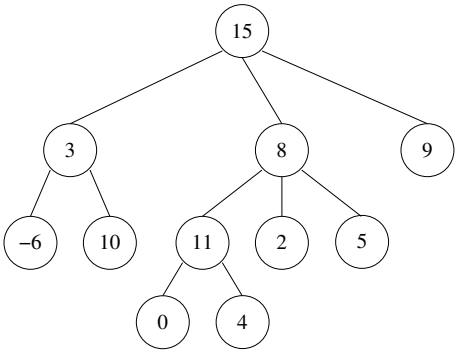


Figure 1 – Arbre général T_1

1 Mesures

Exercice 1.1 (Taille (Size))

1. Donner la définition de la taille d'un arbre.
2. Écrire une fonction qui calcule la taille d'un arbre,
   dans les deux implémentations :
   (a) par n-uplets (chaque nœud contient un n-uplet de fils) ;
   (b) premier fils - frère droit (sous la forme d'un arbre binaire).

Exercice 1.2 (Hauteur (Height))

1. Donner la définition de la hauteur d'un arbre.
2. Écrire une fonction qui calcule la hauteur d'un général, dans les deux implémentations :
   (a) par n-uplets ;
   (b) premier fils - frère droit.

2 Parcours

Exercice 2.1 (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) pour les deux implémentations :
   (a) par n-uplets ;
   (b) premier fils - frère droit.
   Exemple d'utilisation : comment afficher un arbre sous la forme $< oA_1, A_2, ..., A_N >$ avec o le contenu du nœud racine, et A_i les sous-arbres ?

Exercice 2.2 (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 :
   (a) par n-uplets ;
   (b) premier fils - frère droit.

3 Applications

Exercice 3.1 (Représentation *linéaire*)

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 `to_linear` qui construit à partir d'un arbre sa *représentation linéaire* (sous forme de chaîne de caractères), dans les deux implémentations :
   (a) par n-uplets ; (b) premier fils - frère droit.

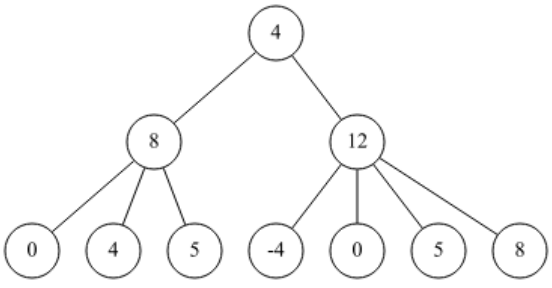
Exercice 3.2 (Le format `dot`)

Un arbre peut être représenté par la liste des liaisons (à la manière d'un graphe) en utilisant le format *dot*.

Avec `T1` l'arbre de la figure 1 :

```
>>> print(tree.dot(T1))
    graph {
        15 -- 3
        15 -- 8
        15 -- 9
        3 -- -6
        3 -- 10
        8 -- 11
        8 -- 2
        8 -- 5
        11 -- 0
        11 -- 4
    }
```

Et s'il y a des nœuds dont les clés sont identiques ?



`trees_examples.py: T3`

Écrire les fonctions qui construisent la représentation au format `.dot` (une chaîne de caractères) d'un arbre dans les deux implémentations, voir `algopy/tree.py` et `algopy/treeasbin.py`.

Pour avoir un aperçu graphique de l'arbre, vous pouvez utiliser "Graphviz" (voir sur gitlab `algopy/readme.md`).

Exercice 3.3 (Famille nombreuse)

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*.

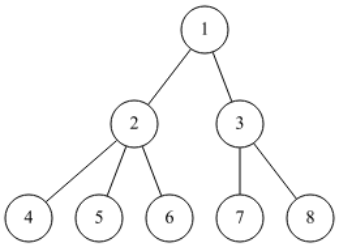


Figure 2 – Arbre `T4`

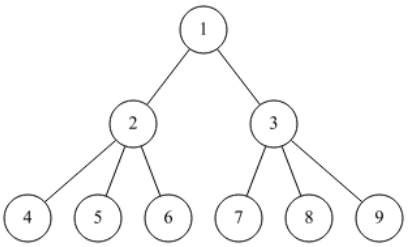


Figure 3 – Arbre `T5`

Exemples d'applications avec les arbres des figures 2 (`T4`) et 3 (`T5`) :

```
>>> morechildren(T4)
False
>>> morechildren(T5)
True
```

Exercice 3.4 (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, dans les deux implémentations :
   (a) par n-uplets ;
   (b) premier fils - frère droit.

Exercice 3.5 (N-uplets ↔ Premier fils - frère droit)

1. Écrire une fonction qui à partir d'un arbre général représenté par un arbre binaire ("Premier fils - frère droit"), construit sa représentation sous forme "classique" (par n-uplets).
2. Écrire la fonction inverse.

Exercice 3.6 (Symétrie – *4 points*)

Écrire la fonction **récursive** `symmetric(`T`,` B`)` qui vérifie si T, un arbre général en implémentation "classique" (par n-uplets) et B, un arbre général en implémentation *premier fils - frère droit*, sont symétriques.

Ci-dessous, les arbres T_6 et T_7 sont symétriques. Les arbres T_6 et T_8 ne le sont pas.

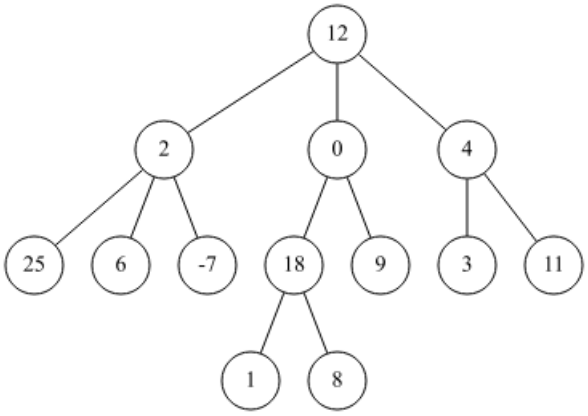


FIGURE 4 – Arbre T_6

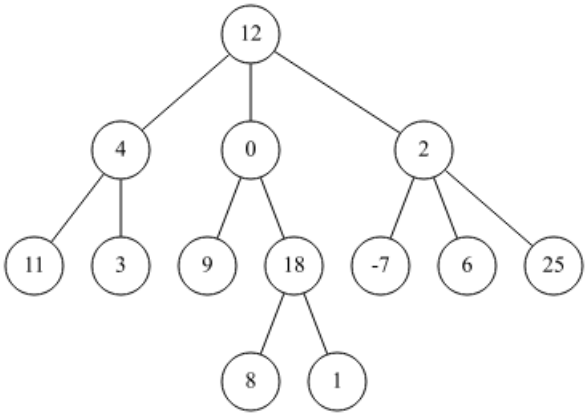


FIGURE 5 – Arbre T_7

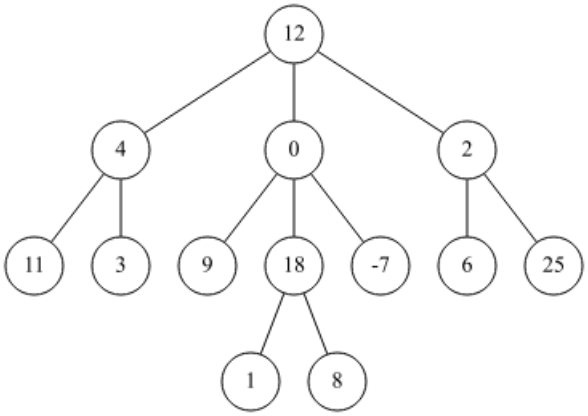


FIGURE 6 – Arbre T_8

Exercice 3.7 (Chargement d'arbres depuis des fichiers)

Pour stocker les arbres dans des fichiers textes (`.tree`) nous utilisons la représentation *linéaire* vue à l'exercice 3.1 ($A =< o, A_1, A_2, ..., A_N >$ est représenté par (o A_1 A_2 ... A_N).)

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