Arbres généraux (General Trees)

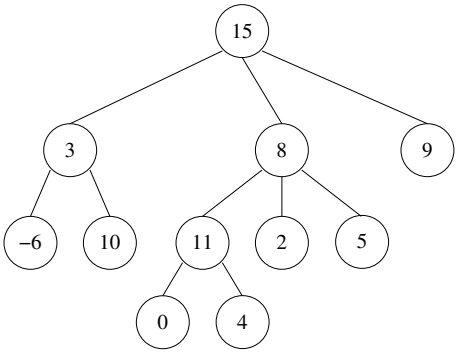

Figure 1 – Arbre général T_1

1 Mesures

Exercice 1.1 (Taille)

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)

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.

Exercice 1.3 (Profondeur moyenne externe)

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

2 Parcours

Exercice 2.1 (Parcours en profondeur)

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.

Exercice 2.2 (Parcours en largeur)

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 calcule la largeur d'un arbre, dans les deux implémentations :
   (a) par n-uplets ;
   (b) premier fils - frère droit.

3 Applications

Exercice 3.1 (Préfixe - Suffixe – *C3 Nov. 2017*)

Dans cet exercice, on se propose de remplir un vecteur avec les clés d'un arbre général. On place chaque clé **deux** fois dans le vecteur : lors de la première rencontre (ordre préfixe) et lors de la *dernière* rencontre (ordre suffixe) du parcours profondeur.

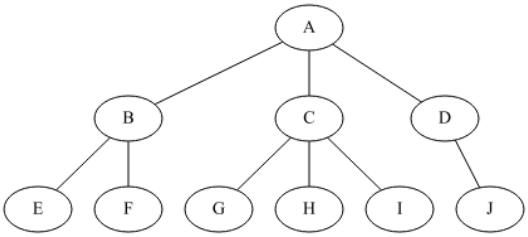

FIGURE 2 – Arbre général T_2

1. Donner le tableau après parcours profondeur de l'arbre de la figure 2.
2. Écrire la fonction `prefsuff(T)` qui, à partir de l'arbre `T`, construit et retourne le vecteur de clés (représenté par une liste en Python) en respectant l'ordre décrit pour les deux implémentations :
   (a) par n-uplets ; (b) premier fils - frère droit.

Exercice 3.2 (Sérialisation – *C3# - April 2017*)

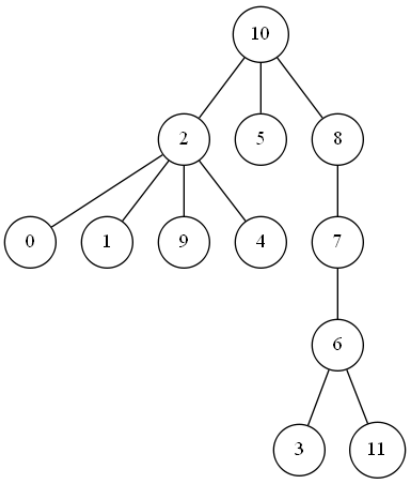

FIGURE 3 – Arbre général T_3

Nous allons nous intéresser à une représentation alternative des arbres généraux : les vecteurs de pères. Cette représentation est linéaire et peut donc être utilisée pour stocker notre arbre dans un fichier (sérialisation).

Le principe est simple : à chaque nœud de l'arbre on associe un identifiant unique, sous la forme d'un entier compris entre 0 et la taille de l'arbre - 1. On construit ensuite un vecteur où la case i contient l'identifiant du père du nœud d'identifiant i. La racine de l'arbre aura pour père -1.

1. Donner le vecteur de pères pour l'arbre de la figure 3
2. Écrire une fonction qui remplit le vecteur (représenté par une liste en Python) de pères correspondant à un arbre pour les deux implémentations :
   (a) par n-uplets ; (b) premier fils - frère droit.

Exercice 3.3 (Le format `dot`)

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

```
graph {
    15 -- 3;
    15 -- 8;
    15 -- 9;
    3 -- -6;
    3 -- 10;
    8 -- 11;
    8 -- 2;
    8 -- 5;
    11 -- 0;
    11 -- 4;
}
```

Autre possibilité :

```
graph {
    15 -- {3; 8; 9};
    3 -- {-6; 10};
    8 -- {11; 2; 5};
    11 -- {0; 4};
}
```

Les ';' peuvent être omis.

Pour avoir un aperçu graphique de l'arbre, vous pouvez utiliser "Graphviz".

Attention : selon l'ordre des liens, le résultat ne sera pas le même. **L'ordre donné ici est celui qui convient pour un arbre.**
Écrire les fonctions qui permettent de créer le fichier `.dot` à partir d'un arbre (dans les deux implémentations).

Exercice 3.4 (Égalité – *C3 - 2016*)

Écrire la fonction `same(`T`,` B`)` qui vérifie si T, un arbre général en représentation "classique" et B, un arbre général en représentation *premier fils - frère droit*, sont identiques : ils contiennent les mêmes valeurs dans les mêmes nœuds.

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 (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), dans les deux implémentations :
   (a) par n-uplets ; (b) premier fils - frère droit.
   Que faut-il modifier pour obtenir une représentation "type abstrait" ($A =< o, A_1, A_2, ..., A_N >$) ?
3. Écrire la fonction réciproque (qui construit l'arbre à partir de la liste) avec les deux implémentations.