

## Arbres généraux (General Trees)

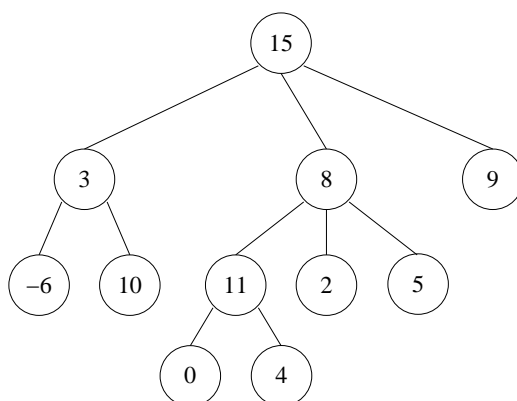


FIGURE 1 – Arbre général  $T_1$

## 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 (Longueur de cheminement externe)

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



## 2 Parcours et tests

### 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 un algorithme qui affiche les clés d'un arbre général niveaux par niveaux, un niveau par ligne, dans les deux implémentations :
  - (a) par  $n$ -uplets ;
  - (b) premier fils - frère droit.

### Exercice 2.3 (Égalité – C3 - Nov. 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.

## 3 Différentes représentations

### Exercice 3.1 (Arité moyenne d'un arbre général – C3 - 2016)

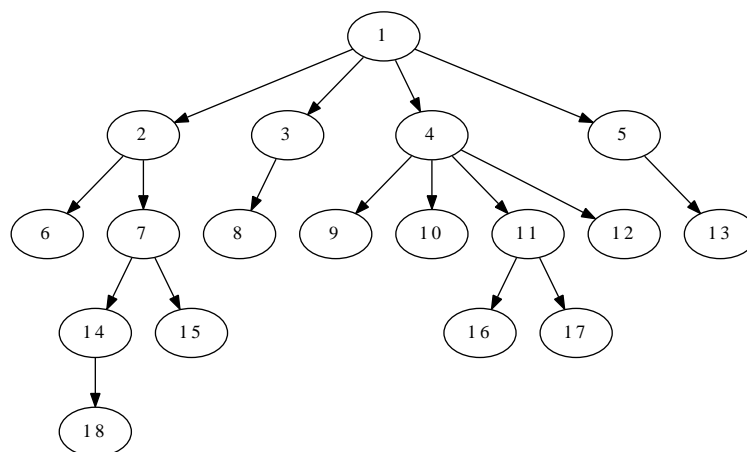


FIGURE 2 – Arbre général

On va s'intéresser à l'arité (nombre de fils d'un nœud) moyenne dans un arbre général. On définit l'arité moyenne comme la somme des nombres de fils par nœud divisée par le nombre de nœuds *internes* (nœuds qui ne sont pas des feuilles).

Par exemple, pour l'arbre de la figure 2, il y a 8 nœuds internes (non feuilles), et lorsque l'on fait la somme des nombres de fils par nœud, on obtient 17 (compter les flèches pour vérifier), l'arité moyenne est donc de  $17/8 = 2.125$ .

Écrire une fonction qui calcule l'arité moyenne d'un arbre général (attention, il ne doit y avoir qu'un seul parcours de l'arbre), dans les deux implémentations :

1. par  $n$ -uplets de pointeurs ;

2. premier fils - frère droit.

**Exercice 3.2 (Sérialisation – Nov. C3 - 2014)**

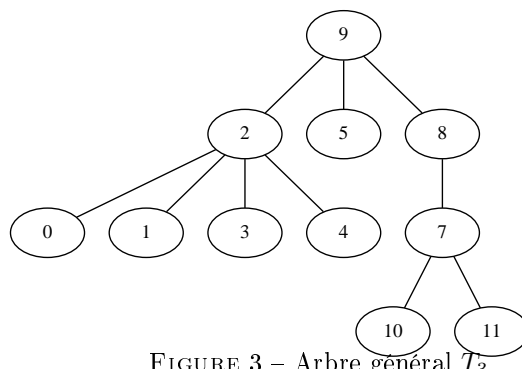


FIGURE 3 – Arbre général  $T_3$

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 (N-uplets  $\leftrightarrow$  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 3.4 (Représentation par listes)**

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

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" ( $A = \langle o, A_1, A_2, \dots, A_N \rangle$ ) ?

**Bonus** Écrire la fonction réciproque (qui construit l'arbre à partir de la liste) avec les deux implémentations.

### Exercice 3.5 (Le format dot)

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

```

1  graph {
2      15 -- 3;
3      15 -- 8;
4      15 -- 9;
5      3 -- -6;
6      3 -- 10;
7      8 -- 11;
8      8 -- 2;
9      8 -- 5;
10     11 -- 0;
11     11 -- 4;
12 }
```

Autre possibilité :

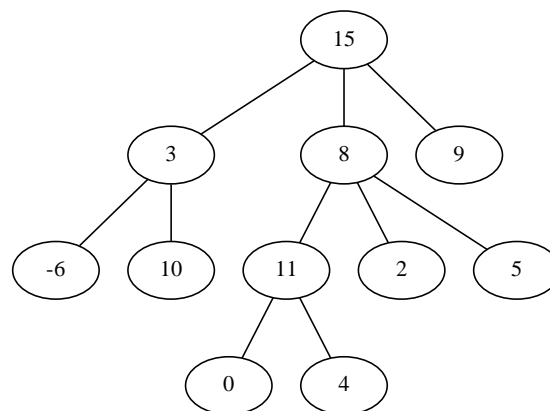
```

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

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)
- et inversement de construire un arbre général (dans les deux implémentations) à partir d'un fichier `.dot`.