Arbres

Activité – Terminale NSI

① Une structure hiérarchique

Exercice 1 *Reconnaître des arbres dans la nature*

Pour chaque situation, indiquer s'il s'agit d'une structure d'arbre. Justifier en une phrase.

1. L'arborescence des fichiers et dossiers de l'ordinateur.
2. Un arbre généalogique des descendants d'une personne.
3. Le réseau social entre les élèves d'une classe (« qui est ami avec qui »).
4. Le sommaire d'un livre (parties, chapitres, sections, sous-sections).
5. Le réseau routier entre plusieurs villes.
6. La structure d'un document HTML.

Remarque 1. Un **arbre** est une structure de données hiérarchique : chaque élément (sauf un, la *racine*) a un *parent unique*. Mathématiquement, un arbre est un graphe **connexe** et **sans cycle**.

② Vocabulaire des arbres

Définition 1 **(Vocabulaire de base)**

— **Nœud** : élément de l'arbre.

— **Racine** : nœud unique au sommet, sans parent.

— **Feuille** : nœud sans enfant.

— **Nœud interne** : nœud qui n'est ni la racine ni une feuille (parfois on inclut la racine).

— **Sous-arbre** : tout nœud, considéré avec ses descendants, forme un arbre.

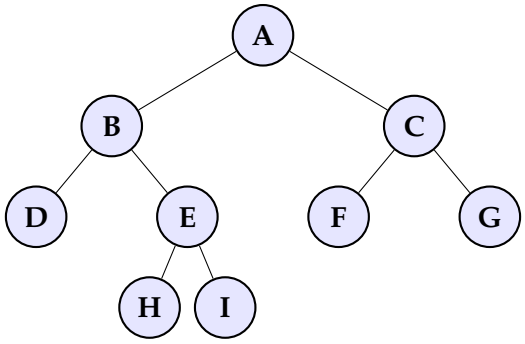
— **Hauteur** d'un nœud : nombre d'arêtes sur le plus long chemin du nœud à une feuille. La hauteur d'une feuille est 0.

— **Profondeur** d'un nœud : nombre d'arêtes du nœud à la racine. La profondeur de la racine est 0.

— **Taille** : nombre total de nœuds dans l'arbre.

Exercice 2 *Lire un arbre*

On considère l'arbre suivant :



1. Quelle est la **racine** de cet arbre ?
2. Quelles sont les **feuilles** ?
3. Quel est le parent de `H` ? Quels sont les enfants de `C` ?
4. Quelle est la **taille** de l'arbre ?
5. Quelle est la **hauteur** de l'arbre (la hauteur de sa racine) ?
6. Quelle est la **profondeur** de `H` ? De `C` ?
7. Combien de nœuds compte le **sous-arbre** enraciné en `B` ?

③ Représenter un arbre en Python

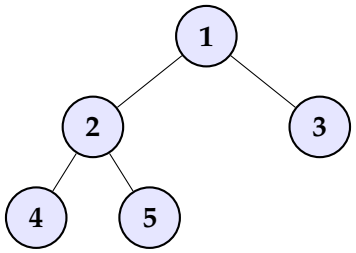
Définition 2 **(Représentation par classe `Noeud`)** On limite ici l'étude aux **arbres binaires** : chaque nœud a au plus deux enfants, appelés *gauche* et *droit*.

```
class Noeud:
    def __init__(self, valeur, gauche=None, droit=None):
        self.valeur = valeur
        self.gauche = gauche
        self.droit = droit
```

Un arbre est représenté par sa racine ; un arbre vide est représenté par `None`. Un sous-arbre absent (gauche ou droit) est aussi `None`.

Exercice 3 *Construire un petit arbre*

On veut construire l'arbre binaire suivant :



1. Compléter le code Python qui construit cet arbre :

```
feuille_4 = Noeud(4)
feuille_5 = Noeud(5)
noeud_2 = Noeud(2, feuille_4, ............)
feuille_3 = ............
racine = Noeud(1, ............, ............)
```

2. Une fois `racine` construit, comment accéder à la valeur de la racine ? À la valeur du fils gauche de la racine ? À la valeur de la feuille `5` ?
3. L'arbre est vide. Quel objet Python le représente ?

④ Algorithmes récursifs sur les arbres

Propriété 1 **(Schéma récursif)** Un arbre est une structure naturellement récursive : chaque sous-arbre est lui-même un arbre. Beaucoup de fonctions sur les arbres se définissent ainsi :

— **cas de base** : l'arbre vide (`None`) ou la feuille ;

— **cas récursif** : combiner les résultats sur le sous-arbre gauche et le sous-arbre droit.

Exercice 4 *Trois fonctions classiques*

1. Compléter la fonction `taille(arbre)` qui renvoie le nombre de nœuds d'un arbre binaire :

```
def taille(arbre):
    if arbre is None:
        return ............
    return ............ + taille(............) + taille(............)
```

2. Compléter la fonction `hauteur(arbre)` qui renvoie la hauteur de l'arbre (par convention, la hauteur d'un arbre vide est -1, celle d'une feuille est 0) :

```
def hauteur(arbre):
    if arbre is None:
        return ............
    h_g = hauteur(arbre.gauche)
    h_d = hauteur(arbre.droit)
    return ............ + max(h_g, h_d)
```

3. Compléter la fonction `nb_feuilles(arbre)` qui compte le nombre de feuilles d'un arbre :

```
def nb_feuilles(arbre):
    if arbre is None:
        return ............
    if arbre.gauche is None and arbre.droit is None:
        return ............
    return nb_feuilles(............) + nb_feuilles(............)
```

5 Les parcours en profondeur

Définition 3 **(Parcours préfixe, infixe, suffixe)** Pour parcourir un arbre binaire en profondeur, on visite chaque nœud à un moment précis :

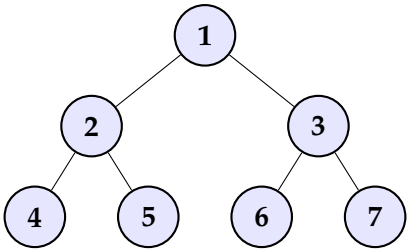
— **préfixe** (NLR) : nœud, puis sous-arbre gauche, puis sous-arbre droit ;

— **infixe** (LNR) : gauche, nœud, droit ;

— **suffixe** (LRN) : gauche, droit, nœud.

Exercice 5 *Trois parcours sur un même arbre*

On considère l'arbre :



1. Donner la liste des nœuds visités dans l'ordre, pour chacun des trois parcours.
2. Compléter la fonction `prefixe(arbre)` qui renvoie la liste des valeurs visitées en parcours préfixe :

```
def prefixe(arbre):
    if arbre is None:
        return ............
    return [arbre.valeur] + prefixe(............) + prefixe(............)
```

3. Adapter pour écrire `infixe` et `suffixe`.

6 Arbres binaires de recherche (ABR)

Définition 4 **(Arbre binaire de recherche)** Un **arbre binaire de recherche** (ABR) est un arbre binaire dans lequel, pour chaque nœud :

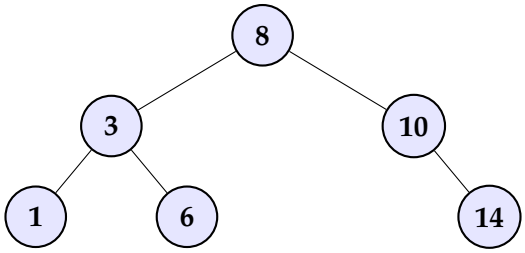
— toutes les valeurs du sous-arbre *gauche* sont **strictement inférieures** à la valeur du nœud ;

— toutes les valeurs du sous-arbre *droit* sont **strictement supérieures** à la valeur du nœud.

Cette propriété permet une recherche très rapide.

Exercice 6 *Vérifier et utiliser un ABR*

On considère l'arbre suivant :



1. Vérifier que cet arbre est bien un ABR.
2. Effectuer le parcours **infixe**. Que constate-t-on ?
3. On cherche la valeur 6. Lister les nœuds visités lors de la recherche.
4. Compléter la fonction `recherche_abr` qui renvoie `True` si `x` est dans l'ABR, `False` sinon :

```
def recherche_abr(arbre, x):
    if arbre is None:
        return ............
    if arbre.valeur == x:
        return ............
    if x < arbre.valeur:
        return recherche_abr(............, x)
    else:
        return recherche_abr(............, x)
```

5. Pourquoi cette recherche est-elle plus rapide que dans une liste non triée ?

⑦ Ce qu'il faut retenir

Propriété 2 **(Vocabulaire)** Un arbre est composé de nœuds reliés par des arêtes, avec une racine unique. Chaque nœud (sauf la racine) a un parent unique. Les nœuds sans enfant sont les feuilles. La taille est le nombre de nœuds ; la hauteur est la longueur du plus long chemin de la racine à une feuille.

Propriété 3 **(Implémentation par classe)** Un arbre binaire est représenté par une classe `Noeud` avec trois attributs : `valeur`, `gauche` (`Noeud` ou `None`), `droit` (`Noeud` ou `None`). L'arbre vide est représenté par `None`.

Propriété 4 **(Schéma récursif)** La plupart des fonctions sur les arbres suivent le schéma : cas de base sur `None`, puis combinaison des résultats sur `arbre.gauche` et `arbre.droit`.

Propriété 5 **(Parcours et ABR)** Trois parcours en profondeur : préfixe (NLR), infixe (LNR), suffixe (LRN). Sur un **ABR**, le parcours infixe donne la liste *triée* des valeurs, et la recherche se fait en $O(\log n)$ si l'arbre est équilibré.