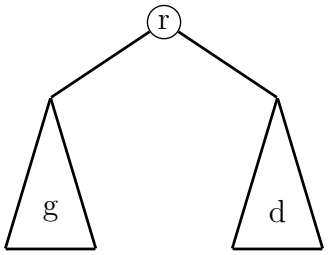
TD n°8

Arbres binaires

Rappel Un arbre binaire A est défini récursivement de la façon suivante :
- soit A est l'arbre vide (celui qui n'a pas de noeuds)
- soit A est défini par un noeud r appelé *racine*, constitué :
  - d'une étiquette (dans ce TD, les étiquettes sont des entiers) ;
  - d'un arbre binaire g appelé *sous-arbre gauche* ;
  - d'un arbre binaire d appelé *sous-arbre droit*.

On peut représenter graphiquement A comme dans la Figure .

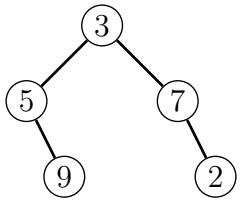


En Java, nous utiliserons les classes suivantes pour représenter un arbre binaire :

```
class Noeud {
    private int etiquette;
    private Arbre gauche;
    private Arbre droit;

    Noeud(int e, Arbre g, Arbre d) {
        this.etiquette = e;
        this.gauche = g;
        this.droit = d;
    }

    public int getEtiquette() {
        return this.etiquette;
    }

    public Arbre getFilsGauche() {
        return this.gauche;
    }
```

```
    public Arbre getFilsDroit() {
       return this.droit;
    }
}


class Arbre {
    private Noeud racine;
    Arbre (Noeud n) {
        racine = n;
    }

    public Noeud getRacine() {
        return this.racine;
    }
}
```

Exercice 1 Créer en Java l'arbre vide, et l'arbre correspondant à l'exemple suivant :



Exercice 2 Parcours
Il existe plusieurs façons de parcourir un arbre. On rappelle ici les trois parcours les plus classiques.

- Parcours **préfixe** :
  1. on visite la racine
  2. on visite le sous-arbre gauche en ordre préfixe
  3. on visite le sous-arbre droit en ordre préfixe
- Parcours **infixe** :
  1. on visite le sous-arbre gauche en ordre infixe
  2. on visite la racine
  3. on visite le sous-arbre droit en ordre infixe
- Parcours **postfixe** :
  1. on visite le sous-arbre gauche en ordre postfixe
  2. on visite le sous-arbre droit en ordre postfixe
  3. on visite la racine

On remarque que ces algorithmes de parcours sont récursifs.

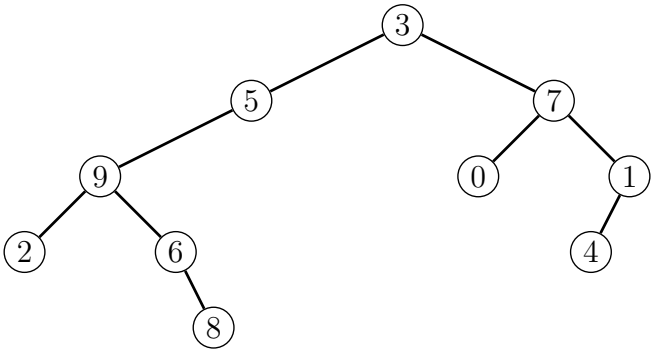


FIG. 1 –

1. Pour l'arbre de la Figure 1, donner la séquence des étiquettes correspondant à un parcours préfixé (de même pour un parcours en ordre infixé et postfixé).
2. Écrire une méthode booléenne `estVide()` qui teste si l'arbre a est vide (c'est-à-dire si sa propriété `racine` contient la valeur `null`).
3. Écrire les méthode `parcoursPrefixe()`, `parcoursInfixe()` et `parcoursPostfixe()` qui affichent les séquences d'étiquettes correspondant aux parcours respectivement préfixé, infixé et postfixé.

Exercice 3 Compter

- Écrire une méthode `nbNoeuds()` qui retourne le nombre de nœuds dans l'arbre.
- Une feuille est un arbre dont la racine contient une valeur et n'a pas de fils. Écrire une méthode `nbFeuilles()` qui retourne le nombre des sous-arbres d'un arbre qui sont de feuilles.

Exercice 4 Recherche

- Écrire une méthode `recherche(int x)` qui cherche et retourne un nœud correspondant à une valeur x passée en paramètre (elle retourne `null` si cette valeur n'apparaît pas dans l'arbre).
- Écrire une méthode `max()` qui retourne un nœud correspondant à la valeur plus grande dans l'arbre, ou `null` si l'arbre est vide.
- On suppose que les valeurs dans l'arbre sont triés de la façon suivante. Pour chaque sous-arbre non vide a :
  - toutes les valeur dans le sous-arbre gauche de a sont plus petites ou égales à la valeur dans la racine de a ;
  - toutes les valeur dans le sous-arbre droit de a sont plus grandes ou égales à la valeur dans la racine de a.

  Écrire deux méthodes `recherche()` et `max()` qui exploitent les arbres avec cette propriété.

Exercice 5 Comparaison

1. Écrire une méthode `egale(Arbre a)` qui prend en argument un arbre `a` et teste s'il est égal à l'arbre courant (même structure et mêmes valeurs).

2. Écrire une méthode `contient(Arbre a)` qui prend en argument un arbre `a` et teste si c'est un sous-arbre de l'arbre courant.

Exercice 6 Profondeur et hauteur
Définitions :

- Un *chemin* dans un arbre a est une séquence de noeuds $n_1, \ldots, n_k$ dans a tel que n_{i+1} est un fils de n_i pour tout $i \in \{1, \ldots, k-1\}$.
- Si $n_1, \ldots, n_k$ est un chemin alors $k-1$ est sa *longueur*.
- La *profondeur* d'un nœud n dans un arbre a est la longueur de l'unique chemin de la racine de a à n. Par example la profondeur de la racine de a est 0, et la profondeur des fils de la racine est 1.
- La *hauteur* d'un arbre a est la profondeur maximum d'un nœud dans a.

Écrire une méthode `hauteur(Arbre a)` qui retourne la hauteur de l'arbre a.