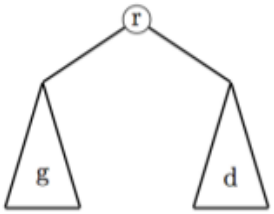
TD n°7

Arbres binaires

1 Rappel sur les arbres

Un arbre binaire A est défini récursivement de la façon suivante :
– soit A est l'arbre vide (il 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 donc représenter graphiquement A comme dans la figure suivante :



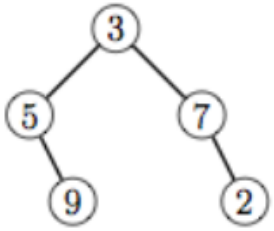
Nous utiliserons la classe `Arbre` pour représenter un arbre binaire :

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

    public Arbre(int etiquette, Arbre gauche, Arbre droit) {
        this.etiquette = etiquette;
        this.gauche = gauche;
        this.droit = droit;
    }

    public int getEtiquette() {
        return etiquette;
    }


    public Arbre getGauche() {
        return gauche;
```

```
    }

    public Arbre getDroit() {
        return droit;
    }
}
```

De même que dans le TD6, toutes les fonctions qui suivent sont statiques pour pouvoir manipuler l'arbre vide (qui est représenté par le pointeur nul).

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



2 Parcours sur les arbres

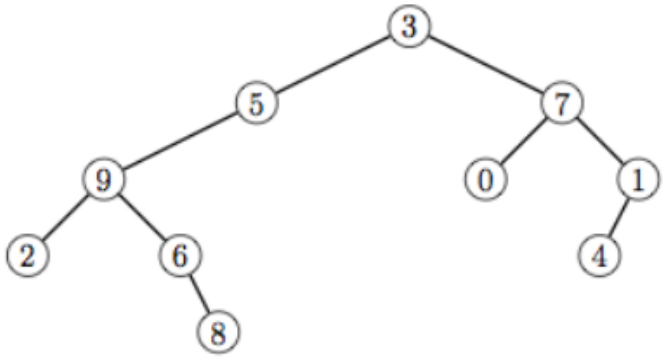
Il existe plusieurs façons de parcourir un arbre. On rappelle ici les trois parcours les plus classiques :

- 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.

Exercice 2 Parcours

1. Pour chacun des trois parcours, donner la séquence des étiquettes visitées de l'arbre suivant :



2. Ajouter les méthodes (statiques) `parcoursPrefixe(Arbre A)`, `parcoursInfixe(Arbre A)` et `parcoursPostfixe(Arbre A)` qui affichent les séquences d'étiquettes correspondant aux parcours respectivement préfixé, infixé et postfixé.

Exercice 3 Compter

1. Ecrire une méthode `int nbNoeuds(Arbre A)` qui retourne le nombre de noeuds dans l'arbre courant.
2. Une feuille est un arbre dont la racine contient une valeur et n'a pas de fils. Ecrire une méthode `int nbFeuilles(Arbre A)` qui retourne le nombre de feuilles de l'arbre courant.

Exercice 4 Recherche

1. Ecrire une méthode `Arbre recherche(Arbre A, int x)` qui retourne le sous-arbre dont la racine est étiquetée par la valeur `x` passée en paramètre (elle retournera `null` si cette valeur n'apparaît pas dans l'arbre).
2. Ecrire une méthode `Arbre max(Arbre A)` qui retourne le sous-arbre dont la racine est étiquetée par la valeur la plus grande dans l'arbre courant, ou `null` si l'arbre est vide.
3. On appelle un arbre binaire de recherche (ABR) un arbre dont les valeurs sont triées de la façon suivante : pour chaque sous-arbre non-vide a :
   – toutes les valeurs dans le sous-arbre gauche de a sont plus petites ou égales à la valeur dans la racine de a,
   – toutes les valeurs dans le sous-arbre droit de a sont plus grandes que la valeur dans la racine de a.

   Ecrire une methode `boolean estABR(Arbre A)` qui teste si l'arbre courant est un ABR.
4. Ecrire les méthodes `Arbre rechercheABR(Arbre A, int x)` et `Arbre maxABR(Arbre A)` pour les ABR.
5. Comparer la complexité des méthodes `recherche` et `rechercheABR` et des méthodes `max` et `maxABR`.

Exercice 5 Comparaison

1. Ecrire une méthode `boolean egale(Arbre A, Arbre B)` qui teste si les arbres `A` et `B` sont égaux (même structure et mêmes étiquettes).
2. Ecrire une méthode `boolean contient(Arbre A, Arbre B)` teste si l'arbre `B` est un sous-arbre de l'arbre `A`.

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 noeud n dans un arbre a est la longeur de l'unique chemin de la racine de a à n. Par exemple, 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 noeud dans a.

Ecrire une méthode `int hauteur(Arbre A)` qui retourne la hauteur de l'arbre `A`.