Exercice 1. Arboriculture : On définit une classe `Arbre` et une classe `Noeud` qui représentent des arbres binaires sans étiquette, de la façon suivante :

```
public class Arbre {
    private Noeud racine;
    public Arbre () { this.racine = null; }
    public Arbre (Noeud n) {
        this.racine = n;
    }
}
public class Noeud {
    private Noeud gauche;
    private Noeud droit;
    // on ne met rien dans les noeuds dans la mesure ou
    // on s'interesse seulement au squelette de l'arbre
    // et pas a son contenu

    public Noeud(Noeud g, Noeud d) {
        this.gauche = g;
        this.droit = d;
    }
}
```

Codez les méthodes suivantes dans `Arbre`, en ajoutant si nécessaire des méthodes dans `Noeud`. Ces méthodes sont illustrées dans les figures 1 à 6 (page suivante).

1. `public void bourgeons()` qui ajoute une génération de feuilles à l'arbre : chaque feuille actuelle de l'arbre `this` va avoir deux nouvelles feuilles comme fils (*i.e.* l'ancienne feuille devient un nœud interne avec deux feuilles comme fils gauche et fils droit). L'arbre vide, reste vide.
2. `public void elagage()` qui supprime toutes les feuilles de `this`. Certains nœuds internes deviennent donc des feuilles. L'arbre qui serait constitué d'une feuille deviendrait vide par cette opération.
3. `public void croissance()`. On appelle *branche* partant d'un noeud une suite de noeuds liés entre eux qui va jusqu'à une feuille. Par choix, le noeud courant ne fait pas partie de la branche. La *longueur de la branche* est son nombre de noeud.

   Partant d'un noeud, on parle de branche gauche ou de branche droite, selon la première direction prise. Un noeud peut ne pas avoir de branche droite ou gauche (voir figure).

   La méthode `croissance()` de l'arbre transforme chacune des branches de longueur 1 de ses noeuds, en les allongeant : une branche gauche de longueur 1 sera remplacée par une branche gauche-gauche de longueur 2, et une branche droite de longueur 1 par une branche droite-droite de longueur 2.
4. `public void decroissance()`. On s'y prend autrement : on va supprimer des étages entiers de l'arbre. On définit la profondeur de la racine comme étant 1, et celles de ses fils comme étant 1 de plus etc.. Pour cette méthode de décroissance on fera en sorte que chaque nœud de profondeur impaire prenne liaison : à gauche au fils gauche de son fils gauche ; et à droite au fils droit de son fils droit. Lorsque ce n'est pas possible il ne se passera rien pour ce nœud. Remarquez que cette décroissance supprime de nombreuses parties (par exemple la partie droite du fils gauche)

5. Modifiez les méthodes précédentes pour qu'elles retournent un entier correspondant respectivement : au nombre de feuilles ajoutées, au nombre de feuilles élaguées, au nombre de noeuds ajoutés et au nombre de noeud supprimés par décroissance (qui n'est pas si trivial). Vous pouvez commencer par coder une méthode `public int nbNoeuds()` qui renvoie le nombre noeuds dans un sous-arbre.

Exercice 2. Chemins : À chaque nœud de l'arbre, on peut associer le `chemin` permettant d'y accéder à partir de la racine. Ce chemin peut être sous la forme "fils gauche, puis fils droit, puis fils droit". Nous les représenterons sous la forme d'une chaîne de caractères de 'g' et de 'd' : "gdd" par exemple.

Codez les méthodes suivantes dans `Arbre`, en ajoutant si nécessaire des méthodes dans `Noeud`

1. `public Arbre sousArbre(String chemin)` qui retourne le sous-arbre de `this` qui est enraciné sur le nœud dont le chemin est indiqué par `chemin`. Si le chemin est invalide on retournera l'arbre vide (par exemple pour "gg" ou "gdgd" sur l'arbre `a` de la figure 1). La méthode intermédiaire correspondante dans la classe `Noeud` aura la signature `public Noeud sousArbre(String chemin)`. Si ce chemin n'est pas dans l'arbre, par exemple "gg" ou "gdgd" pour le dessin de l'arbre `a`, la méthode retournera `null`.
2. `public void greffe(Arbre a, String chemin)` qui greffe l'arbre `a` dans l'arbre courant ; il prendra la place du nœud repéré par le `chemin` donné, c'est à dire que la racine de `a` s'insère à l'emplacement correspondant à la dernière lettre du chemin. Si le chemin est trop long ou incorrect, on ne fait rien (en particulier l'arbre vide restera toujours vide).
3. `public void echange(String chemin1, String chemin2)` qui échange les sous-arbres implantés aux chemins donnés en argument (si l'un des chemins n'est pas valide, on ne fait rien).

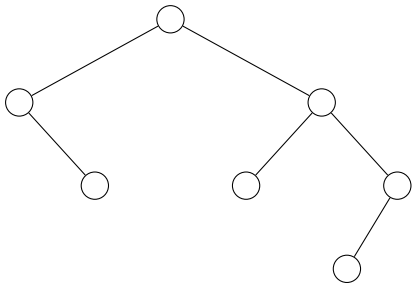

FIGURE 1 – `Arbre a`

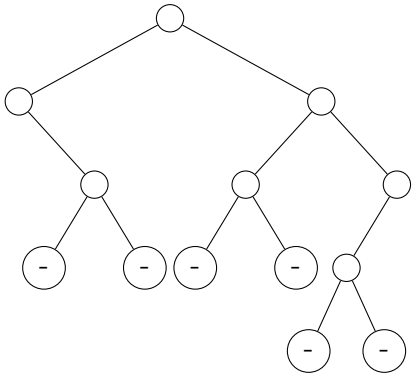

FIGURE 2 – `Arbre b = a.bourgeons()`, les nœuds ajoutés sont indiqués par une "-"

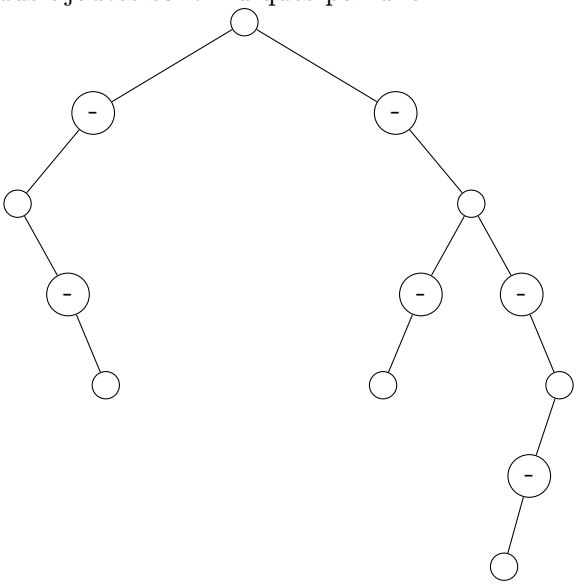

FIGURE 4 – `Arbre d = a.croissance()`, les nœuds ajoutés sont indiqués par une "-"

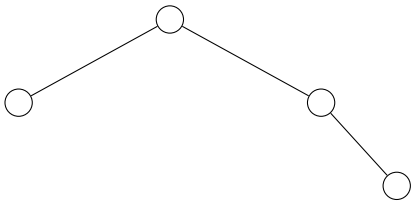

FIGURE 3 – `Arbre c = a.elagage()`

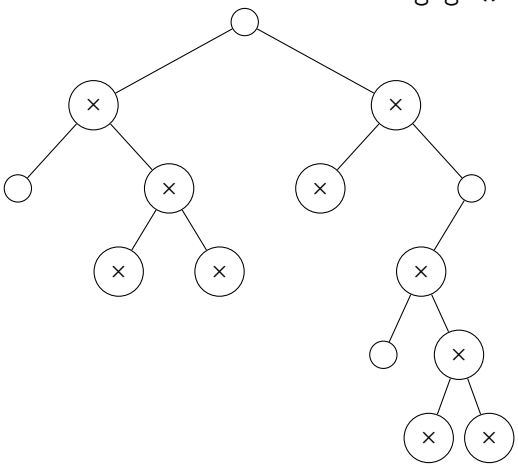

FIGURE 5 – `Arbre e`, les nœuds qui seront supprimés lors de la décroissance sont indiqués par un "×"

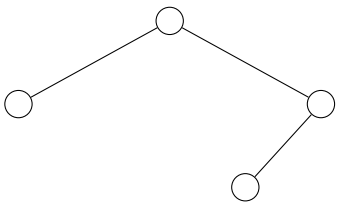

FIGURE 6 – `Arbre f = e.decroissance()`