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;

    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 à chaque feuille de l'arbre `this` va ajouter deux feuilles comme fils, *i.e.* chaque 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 réduit à une feuille devient vide.
3. `public void croissance()` qui transforme chaque branche gauche de longueur 1 par une branche gauche de longueur 2 et chaque branche droite de longueur 1 par une branche droite de longueur 2.
4. `public void decroissance()` est la méthode inverse. Chaque nœud de niveau pair est relié à gauche au fils gauche de son fils gauche et à droite au fils droit de son fils droit.
5. *(Facultatif)* 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. Attention : pour `decroissance`, il ne faut pas oublier de compter les noeuds des branches qui ont été supprimées (par exemple la branche droite du fils gauche) ; vous pourrez commencer par coder une méthode `public int nbNoeuds()` qui renvoie le nombre noeud 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" pour l'exemple précédent.
Codez les méthodes suivantes dans `Arbre`, en ajoutant si nécessaire des méthodes dans `Noeud`

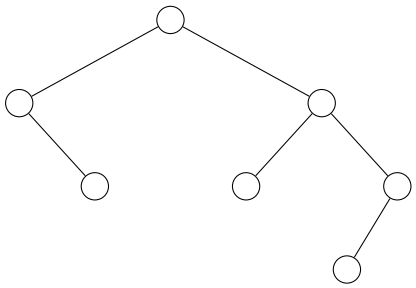

FIGURE 1 – `Arbre a`

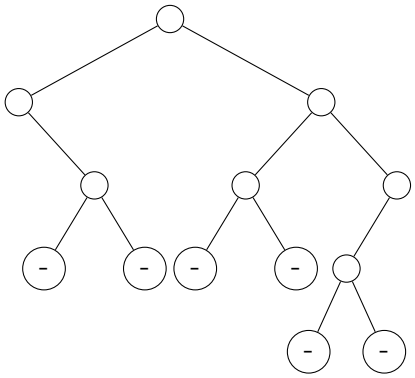

FIGURE 2 – `Arbre b = a.bourgeons()`, les nœuds ajoutés sont indiqués par une "-"

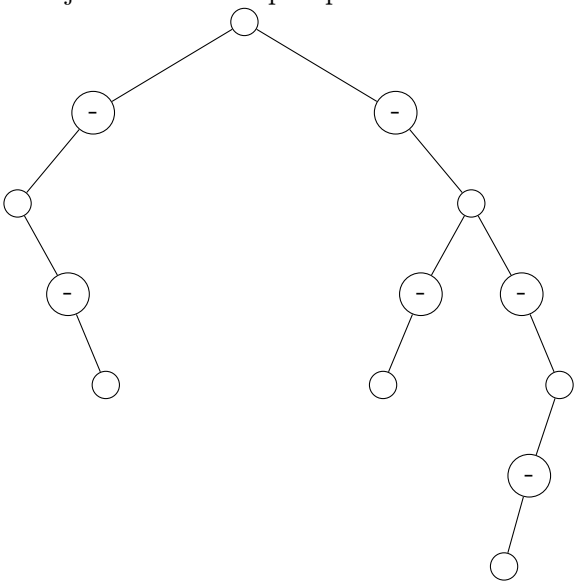

FIGURE 4 – `Arbre d = a.croissance()`, les nœuds ajoutés sont indiqués par une "-"

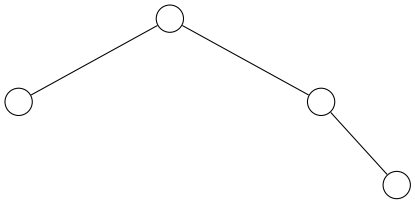

FIGURE 3 – `Arbre c = a.elagage()`

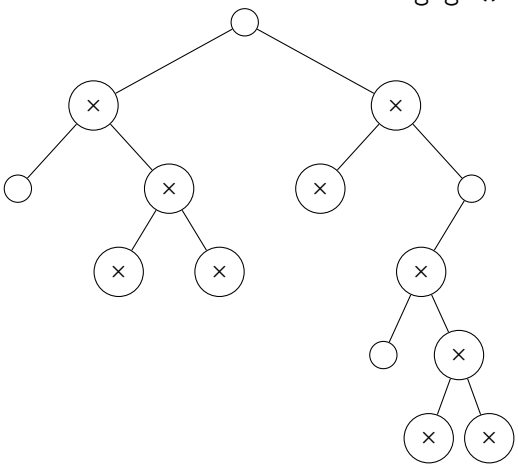

FIGURE 5 – `Arbre e`, les nœuds qui seront supprimés lors de la décroissance sont indiqués par un "×"

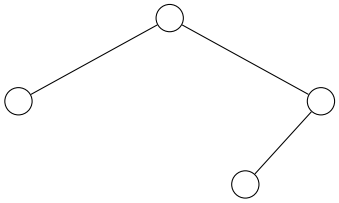

FIGURE 6 – `Arbre f = e.decroissance()`

1. `public Arbre sousArbre(String chemin)` qui retourne le sous-arbre de `this` qui est situé sur le noeud dont le chemin est indiqué par `chemin`. Si le chemin sort de l'arbre on retournera l'arbre vide. Pour cela, dans `Noeud`, on codera une méthode `public Noeud sousArbre(String chemin)`, qui retourne le sous-arbre commençant à `chemin`. Si ce chemin n'est pas dans l'arbre, par exemple "gg" ou "gdgd" pour le dessin de l'arbre `a`, la méthode retourne `null`.
2. `public void greffe(Arbre a, String chemin)` qui greffe l'arbre `a` sur le nœud de chemin `chemin`, 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 reste vide sauf si `chemin` est la vide et `a` ne l'est pas).
3. *(Facultatif)* `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).