```
 if (a == null) return new Arbre(x,null,null);
 if (x < a.val) {a.gauche = insert(x,a.gauche); return a;}
 if (x > a.val) {a.droit = insert(x,a.droit); return a;}
 return a;}
```

La suppression d'un élément de l'arbre demande également un temps proportionnel à la hauteur de l'arbre, mais la méthode est un peu moins simple que la recherche ou l'insertion. Une fois le nœud à supprimer trouvé, il faut distinguer trois cas selon le nombre d'enfants de ce nœud. Si c'est une feuille, il suffit de le supprimer. S'il a un seul enfant, il suffit de le supprimer et de faire remonter son sous-arbre d'un niveau. S'il en a deux, alors il faut rechercher le plus grand élément de son sous-arbre gauche, remplacer le nœud supprimé par cet élément et supprimer récursivement cet élément dans le sous-arbre gauche.

On peut remarquer que la suppression de cet élément dans le sous-arbre gauche se fera nécessairement en une seule étape, car le plus grand élément d'un arbre ne peut pas avoir d'enfant droit.

```
static int max (final Arbre a) {
 if (a.droit == null) return a.val; else return (max(a.droit));}

static Arbre suppress (final int x, final Arbre a) {
 if (a == null) return null;
 if (x == a.val) {
  if ((a.gauche == null) && (a.droit == null)) return null;
  else if (a.gauche == null) return a.droit;
  else if (a.droit == null) return a.gauche;
  else {int m = max(a.gauche);
        a.val = m;
        a.gauche = suppress(m,a.gauche);
        return a;}}
 if (x < a.val) {a.gauche = suppress(x,a.gauche);return a;}
 a.droit = suppress(x,a.droit);return a;}
```

Exercice 9.4

Pour pouvoir programmer ces fonctions par des méthodes dynamiques et que ces méthodes puissent modifier leur argument y compris quand c'est l'arbre vide, une solution est d'envelopper le type `Arbre`.

Réécrire ces trois méthodes pour le type des arbres enveloppé.