Algorithmique
TD 9. Semaine du 10/12/2000

On considère la structure d'arbre binaire définie par la classe `Arbre` du TD 8 :

```
class Arbre{
    Noeud racine;
}
```

```
class Noeud{
      int cle;
      Noeud gauche, droit;
}
```

Exercice 1

On appelle *arbre binaire de recherche* (ABR) tout arbre binaire vérifiant la propriété suivante: si un nœud est étiquetée par n alors,

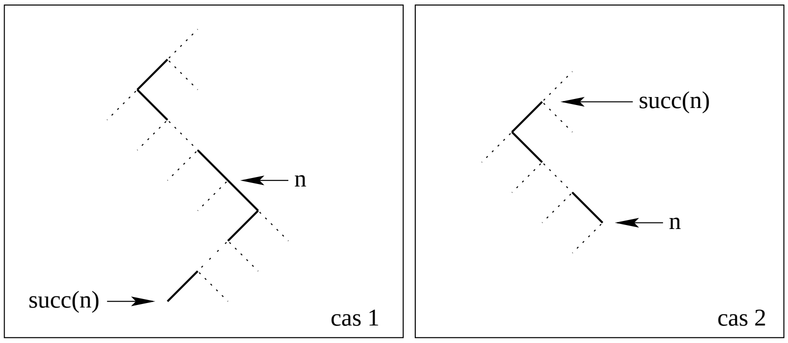
- tous ses descendants à gauche sont étiquetés par des entiers strictement plus petits que n,
- tous ses descendants à droite sont étiquetés par des entiers strictement plus grands que n.

1) Implémenter la méthode `void ajouter(int m)` prenant en arguments un entier `m`, et ajoutant une feuille étiquetée par m si m n'est pas déjà dans l'arbre. L'arbre devra rester un ABR.

2) Implémenter la méthode `int minimum()` qui retourne la valeur de la plus petite étiquette. On remarquera que ce minimum est l'étiquette du nœud le plus à gauche de l'arbre.

Exercice 2

Etant donné un noeud arbitraire d'un ABR étiqueté par n, le *successeur* de ce nœud est le nœud étiqueté par la plus petite étiquette plus grande que n.



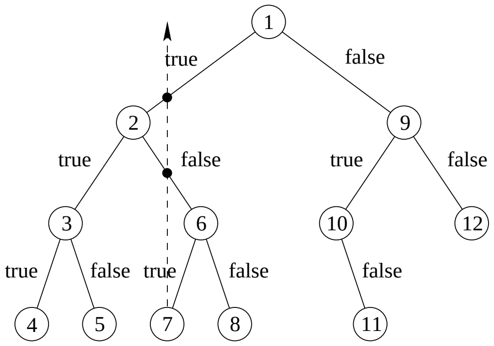
1) Montrer que si un nœud a un descendant droit, son successeur, s'il existe, est le minimum de son sous-arbre droit.

2) Montrer que si un nœud n'a pas de descendant droit, son successeur, s'il existe, est le premier de ses ancêtres dont le fils gauche est aussi l'un de ses ancêtres.

3) Implémenter la méthode `int successeur(int n)`.

Exercice 3

On souhaite écrire en `Java` un afficheur `ascii` pour la classe `Arbre`, représen-tant les branches à l'aide des caractères "`-`" et "`|`". Les etiquettes de l'arbre et les "`|`" seront placées sur les colonnes atteintes par "`\t`". L'afficheur doit fonctionner pour tout arbre dont les étiquettes font au plus deux chiffres.




```
                12 ----|
        9 ---- |
        |       |       11 ----|
        |       10 ----|
1 ---- |
        |
        |               8 ---- |
        |       6 ---- |
        |       |       7 ---- |
        2 ---- |
                |       5 ---- |
                3 ---- |
                        4 ---- |
```

Dans la figure de gauche, les nœuds de l'arbre sont placés sur un quadrillage. La distance séparant deux nœuds de même niveau est le double de celle du niveau inférieur. La position d'un nœud est décrite par une suite de booléens. On dira que le noeud n *voit* l'arc a, si la droite verticale partant du centre de `n` traverse a - par exemple, le nœud 7 voit les arcs (2,6) et (1,2).

1. Soit n un nœud relié à la racine par la suite d'arcs $(a_1, \ldots a_p)$, de position $(b_1, \ldots, b_p)$.
   (a) quels sont, en fonction de la position de n, les arcs vus par n ?
   (b) soit n' un fils de n, de position $(b_1, \ldots, b_p, b_{p+1})$. Quels sont les arcs vus par n' ?
   (c) appelons *prefixe* de n la suite de "`|`" et de "`\t`" précédant son étiquette dans la ligne `ascii` affichant n.
   Ecrire le pseudo-code construisant, à partir du préfixe de n et de b_p, le préfixe des deux fils de n. On distinguera les cas $p = 0$ (le nœud n est au sommet de l'arbre) et $p > 0$.

2. En déduire le code de

   ```
   void affiche(String prefixe, boolean b, boolean sommet)
   ```

   tel que `a.affiche("", true, true)` affiche `a` comme ci-dessus.