TP n°7

Arbres binaires de recherche

Rappels

Un arbre binaire de recherche est un arbre binaire tel que chacun de ses noeuds est etiquete par une valeur n qui veri e :
{ Tous les noeuds de son sous-arbre gauche ont une valeur inferieure a n.
{ Tous les noeuds de son sous-arbre droit ont une valeur superieure a n.
Pour eviter les cas particuliers dans cette presentation, on ne considerera ici que des arbres binaires de recherche qui ne contiennent pas deux fois la même valeur.

Nous rappelons la facon de modeliser les arbres binaires que nous avons choisi :

```
public class Noeud {
 private int etiquette;
 private Arbre gauche; // fils gauche du noeud
 private Arbre droit;  // fils droit du noeud

 Noeud (int e){
  etiquette = e;
  gauche = new Arbre();
  droit = new Arbre ();
 }
}

public class Arbre {
 Noeud racine;

 Arbre (){
  racine = null;
 }
}
```

On suppose en outre que l'on dispose des accesseurs et des modi eurs attendus.

Toutes les methodes demandees devront être des methodes dynamique de la classe Arbre. On supposera, sans le veri er, que les arbres respectent bien la propriete d'être des arbres binaires de recherche.

Exercice 1 Ecrire une methode profondeur qui renvoie la profondeur de l'arbre courant.

Ajout d'un element

Lorsqu'on cherche a inserer un element dans un arbre binaire de recherche, il faut trouver la position ou il viendra se loger. Suivant la defnition recursive d'un arbre binaire de recherche, on partira de la racine et :

- Si l'arbre est vide (cas de base) : on renvoie un arbre constitue d'un seul noeud qui contient la valeur a inserer.
- Sinon, on doit faire face a trois cas possibles :
  - L'element a inserer est inferieur a la valeur de l'etiquette courante, dans ce cas on repete la methode pour inserer le nouvel element dans son sous-arbre gauche.
  - L'element a inserer est superieur a la valeur de l'etiquette courante, dans ce cas on repete la methode pour inserer le nouvel element dans son sous-arbre droit.
  - L'element est egal a la valeur de l'etiquette courante, dans ce cas on ne fait rien (rappelez vous : on ne traite ici que les arbres binaires de recherche qui ne contiennent pas deux fois la même valeur).

Exercice 2 Ecrire une methode `inserer_dans_abr` qui prend en argument un entier et qui l'ajoute a l'arbre courant. Tester votre methode avec plusieurs insertions, et en a chant l'arbre dans l'ordre pre xe, in xe et su xe, que remarquez vous ?

Exercice 3 Ecrire une methode `est_dans_abr` qui prend en argument un entier et qui renvoie `true` si l'entier est dans l'arbre courant et `false` dans le cas contraire.

Exercice 4 Ecrire une fonction `max_abr` qui renvoie la valeur du plus grand element de l'arbre courant qu'on supposera non-vide.

Exercice 5 Ecrire une fonction `min_abr` qui renvoie la valeur du plus petit element de l'arbre courant qu'on supposera non-vide.

Exercice 6 Ecrire une methode `est_abr` qui teste si l'arbre courant est un arbre binaire de recherche. (Remarque que c'est cette methode qui justi e qu'on puisse par ailleurs supposer que les arbres qu'on etudie sont bien des ABRs)

Suppression d'un element

Pour supprimer un noeud d'un arbre binaire de recherche, on peut commencer par deux cas simples :

- Soit le noeud est une feuille, dans ce cas on le supprime tout simplement. (Remarquez le rapport au pere)
- Soit le noeud n'a qu'un seul ls, dans ce cas on remplace ce noeud par ce ls.

Dans le cas restant, pour supprimer un noeud qui a deux ls, on remplacera la valeur de ce noeud par la valeur du plus grand element de son sous-arbre gauche que l'on supprimera ensuite. (On aurait pu tout aussi bien remplacer sa valeur par le plus petit element de son sous-arbre droit).

Exercice 7 Ecrire une methode `supprimer_max_abr` qui renvoie l'arbre courant prive de son element maximal. Cette methode sera utilisee dans l'exercice suivant.

Exercice 8 Ecrire une methode `supprimer_dans_abr` qui prend en argument un entier, et qui renvoie l'arbre courant prive de cet entier, en distinguant selon les trois cas que nous venons d'etudier.