IF122 Feuille de TD/TP 10 : Arbres binaires de recherche

Rappels.

Un arbre binaire de recherche est un arbre binaire dans lequel chaque noeud qui est étiqueté par une valeur n est tel que :

- Tous les noeuds de son sous–arbre gauche ont une valeur inférieure à n.
- Tous les noeuds de son sous–arbre droit ont une valeur supérieure à n.

Pour éviter les cas particuliers dans cette présentation on ne considérera ici que des arbres binaires de recherche qui ne contiennent pas deux fois la même valeur.

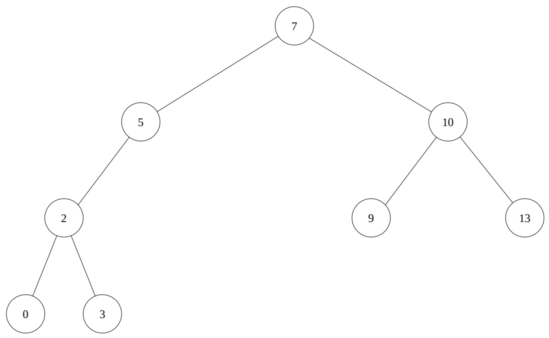

Fig. 1 – Un arbre binaire de recherche.

Nous rappelons la façon de modéliser les arbres binaires que nous avons choisie :

```
Class Arbre {
  Noeud n;

 Arbre (){...}
 Arbre (int x, Arbre a, Arbre b){ ...}
 public static boolean est_vide(Arbre a) { ...}
 public static int get_etiquette(Arbre a){ ...}  // pour une version statique
 public static Arbre fils_gauche(Arbre a){ ...}
 public static Arbre fils_droit (Arbre a){ ...}
}

Class noeud
{
  int val;
  Arbre fg; // fils gauche du noeud
  Arbre fd; // fils droit du noeud
}
```

▸ **Exercice 1 (TD).** *Rappeler la procédure de parcours d'un arbre binaire en ordre infixe et donnez le résultat de l'application de cette procédure à l'arbre de la figure 1.*

▸ **Exercice 2 (TD).** *Écrire une méthode* est_dans_abr *qui prend en argument un arbre binaire de recherche et un entier et qui renvoie* true *si l'entier est dans l'arbre et* false *dans le cas contraire.*

▸ **Exercice 3 (TD).** *Comment trouver le plus grand élément d'un arbre binaire de recherche ? Écrire une fonction* max_abr *qui prend en argument un arbre binaire non vide de recherche et qui renvoie la valeur de son plus grand élément.*

▸ **Exercice 4 (TP).** *Comment trouver le plus petit ?*

Ajout d'un élément.

Lorsqu'on cherche à insérer un élément dans un arbre binaire de recherche, il faut trouver la position où il viendra se loger. Suivant la définition récursive d'un arbre binaire de recherche, on partira de la racine et :

- Si l'arbre est vide (cas de base) : on renvoie un arbre constitué d'un seul noeud qui contient la valeur à insérer.
- Sinon on doit faire face à trois cas possibles :
  - L'élément à insérer est inférieur à la valeur de l'étiquette courante, dans ce cas on répète la méthode pour insérer le nouvel élément dans son sous–arbre gauche.

– L'élément à insérer est supérieur à la valeur de l'étiquette courante, dans ce cas on répète la méthode pour insérer le nouvel élément dans son sous–arbre droit.
– L'élément est égal à la valeur de l'étiquette 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 5 (TD).** *Donner le résultat de l'insertion des éléments 8, 6 et 4 dans l'arbre binaire représenté par la figure 1.*

▸ **Exercice 6 (TD).** *Remarquez qu'étant donné un ensemble d'étiquettes, l'arbre binaire de recherche n'est pas unique.*

▸ **Exercice 7 (TP).** *Écrire une méthode* inserer_dans_abr *qui prend en argument un arbre binaire de recherche et un entier et qui retourne l'arbre binaire de recherche résultat.*

Suppression d'un élément

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.
– Soit le noeud n'a qu'un seul fils, dans ce cas on remplace ce noeud par son fils.

▸ **Exercice 8 (TD).** *Écrire une méthode* supprimer_max_abr *qui prend en argument un arbre binaire de recherche et qui renvoie l'arbre de départ privé de son élément maximal.*

Pour supprimer un noeud qui a deux fils, on remplace la valeur de ce noeud par la valeur du plus grand élément de son sous–arbre gauche que l'on supprimera ensuite. (On aurait pu tout aussi bien remplacer sa valeur par le plus petit élément de son sous–arbre droit).

▸ **Exercice 9 (TD).** *Écrire une méthode* supprimer_dans_abr *qui prend en argument un arbre binaire de recherche et un entier, et qui renvoie cet arbre privé de cet entier.*

Correction

Exercice 2

```
public static boolean est_dans_abr(Arbre a, int x){
   if (est_vide(a)) return false;
   int v= get_etiquette(a);
   if (v==x) return true;
   if (x > v) return est_dans_abr(fils_droit(a),x);
   return est_dans_abr(fils_gauche(a),x);
}
```

Exercice 3

```
public static int max_abr(arbre a){
   Arbre d=fils_droit(a);
   if (est_vide(d)) return get_etiquette(a);
   else return max_abr(d);
}

// vous pouvez faire le min de facon iterative pour changer ...
```

Exercice 7

```
public static Arbre inserer_dans_abr(Arbre a, int x) {
   if (est_vide(a)) a.n=new Noeud(x);
   int e=get_etiquette(a);
   if ( x < e ) {
       Arbre b=inserer_dans_abr(fils_gauche(a),x);
       a.set_fils_gauche(b);  // triviale, a faire statique ?
       return a;
   }
   if (  e < x ) {
       Arbre b=inserer_dans_abr(fils_droit(a),x);
       a.set_fils_droit(b);
       return a;
   }

   return a; // cas ou x est deja present, on nen change rien
}
```

Exercice 9

```
public static void suppression_simple(Arbre a,int x){
   if est_vide(a) return;
   Arbre g=fils_gauche(a);
   Arbre d=fils_droit(a);
   int e=get_etiquette(a);

   if (x<e) {suppression_simple(g,x); return;}
```

```
   if (x>e) {suppression_simple(g,x);return;}

   if (! (est_vide(g) || est_vide(d)))  return; // pas simple, on ne
                                                // sait pas faire
   if (est_vide(g) && est_vide(d)) {
       a.n= null;
       return;
   }

   if (est_vide (g)) {a.n=d; return;}
   a.n=g; return;
}

public static void supprimer_dans_abr(Arbre a,int x){
   if (est_vide(a)) return; // pas trouve

   Arbre g=fils_gauche(a);
   Arbre d=fils_droit(a);
   int e=get_etiquette(a);

   if (x<e) {supprimer_dans_abr(g,x); return;}
   if (x>e) {supprimer_dans_abr(d,x);return;}

   if (est_vide(g) || est_vide(d)) {
       suppression_simple(a,x);
       return;
   }

   int m=max_abr(g);
   a.set_etiquette(m);

   suppression_simple(g,m);
}
```