UE Programmation Orientée Objet
Examen Seconde session – Juin 2005

3 heures - notes et polycopiés de cours/TD/TP autorisés
livres et calculatrices interdits

Le code Java devra être compatible avec la version 1.4 du Jdk.
Comme toujours il est conseillé de lire une fois le sujet en entier avant de commencer à le traiter.

ARBRES BINAIRES

La structure de données arbre binaire est définie récursivement ainsi :

- ▷ l'arbre vide est un arbre binaire,
- ▷ un arbre binaire qui n'est pas l'arbre vide est un arbre défini par une racine contenant une valeur et qui a **exactement** deux fils, appelés fils gauche et fils droit, qui sont des arbres binaires. Un arbre binaire non vide est donc un triplet défini par la valeur de sa racine et ses deux sous-arbres.

On appelle *feuille* un arbre binaire réduit à un seul nœud.
On appelle *taille* de l'arbre, son nombre de nœuds. La taille d'un arbre vide est 0.
On appelle *hauteur* de l'arbre, la longueur de sa plus longue branche (du nœud racine à un nœud feuille). La hauteur d'une feuille est 0 et la hauteur d'un arbre vide est -1.
Voici quelques exemples d'arbres binaires (l'arbre vide est noté △) :

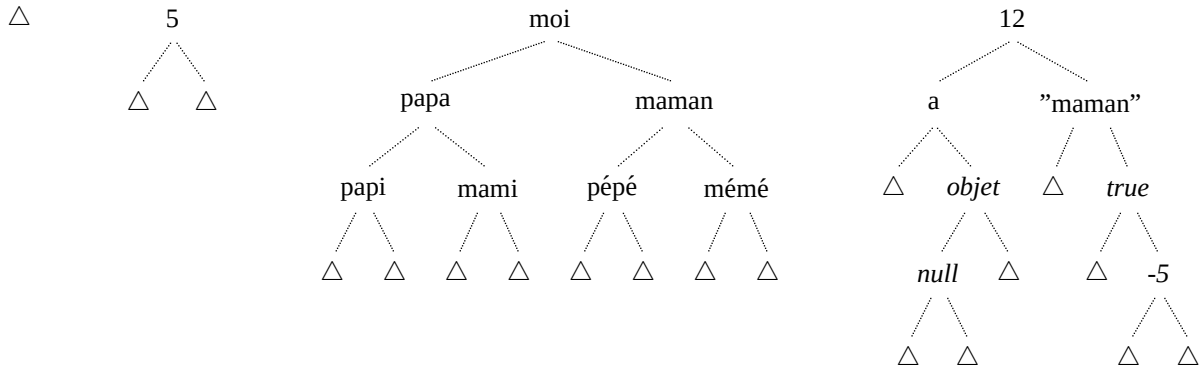


Figure 1: Quatre exemples d'arbres binaires, de gauche à droite : l'arbre vide, un arbre réduit à une feuille puis deux arbres avec des valeurs quelconques aux nœuds. Ces arbres sont respectivement de taille : 0, 1, 7 et 7. Leurs hauteurs respectives sont : -1, 0, 2 et 3.

Q 1 . Une interface définissant les arbres binaires est fournie en annexe : `AB`. Donnez le code Java d'une classe `ArbreBinaire` qui implémente cette interface et permet de définir des arbres binaires.

Un arbre vide sera défini par un arbre binaire dont les deux sous-arbres ont pour valeur `null` (NB : ce n'est pas le cas d'une feuille !).

Q 2 . Il existe plusieurs manières de parcourir les différents nœuds d'un arbre binaire afin d'en énumérer les valeurs. On peut parler dans ce cas de "visiter" l'arbre binaire. Les visiteurs les plus connus sont le visiteur en "largeur d'abord" et le visiteur en "profondeur d'abord" (ils sont expliqués un peu plus loin).

Vous trouverez en annexe l'interface `ABVisiteur` qui définit cette notion. Cette interface ne propose qu'une méthode dont la fonction et de renvoyer un itérateur sur les **valeurs** des nœuds de l'arbre qui permet de parcourir ces valeurs selon un ordre défini par le visiteur.

Q 2.1. Le visiteur en "largeur d'abord".
Il est nommé ainsi car il propose un parcours des nœuds de l'arbre "niveau par niveau". C'est-à-dire que tous les nœuds situés à une profondeur donnée sont parcourus avant les nœuds situés plus profondément dans l'arbre. A une profondeur donnée, les nœuds sont examinés de gauche à droite. La figure 2 présente un exemple de parcours en largeur d'abord sur un arbre binaire.
Donnez le code `Java` d'une classe `VisiteurABLargeur` qui réalise un tel visiteur. On peut remarquer que parcourir en largeur d'abord un arbre binaire revient à gérer une file à laquelle on ajoute, quand ils ne sont pas vides, les valeurs des fils gauche et droit du premier élément de la file que l'on supprime avant de la file. Le parcours est terminé quand la file est vide.
Attention : ce sont les nœuds qui sont parcourus, mais ce sont les valeurs de ces nœuds que l'on souhaite pouvoir récupérer.

Q 2.2. Le visiteur en “profondeur d’abord”.

Il est nommé ainsi car il propose un parcours des nœuds de l’arbre en descendant dans l’arbre en priorité le plus profondément et le plus à gauche possible. Lorsqu’une feuille a été atteinte c’est ensuite le nœud qui avait été laissé de côté le plus récemment lors de la descente dans l’arbre qui est considéré. La figure 2 présente un exemple de parcours en profondeur d’abord sur un arbre binaire.

Donnez le code `Java` d’une classe `VisiteurABProfondeur` qui réalise un tel visiteur. On peut remarquer que parcourir en profondeur d’abord un arbre binaire revient à gérer une pile à laquelle on ajoute (en tête donc), quand ils ne sont pas vides, les valeurs des fils droit puis gauche du premier élément de la pile que l’on supprime préalablement. Le parcours est terminé quand la pile est vide.

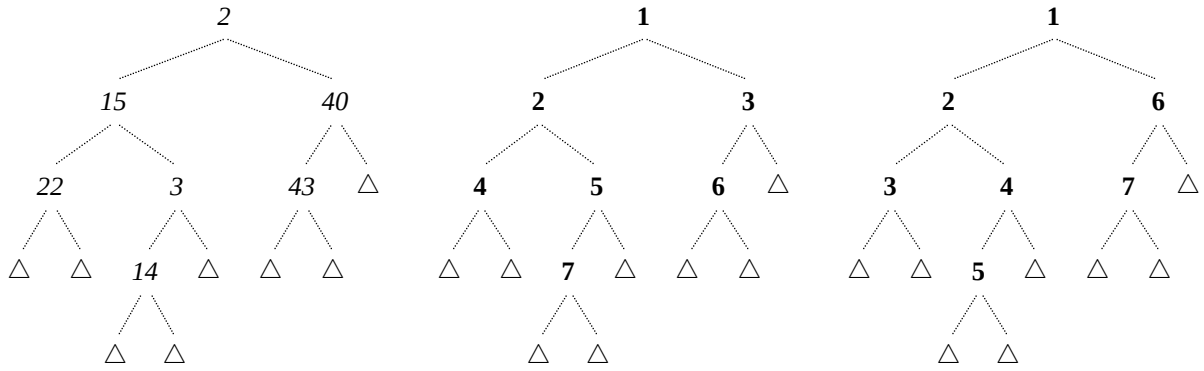
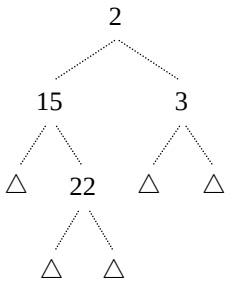


Figure 2: Parcours d’arbre, à gauche l’arbre binaire avec les valeurs des différents nœuds. Au centre, parcours en largeur d’abord, les nœuds sont numérotés selon leur ordre de parcours, chaque niveau de profondeur est examiné avant le suivant, les valeurs des nœuds sont donc examinées dans l’ordre : $\{2,15,40,22,3,43,14\}$. A droite, parcours en profondeur d’abord, les nœuds sont numérotés selon leur ordre de parcours, les valeurs des nœuds sont donc examinées dans l’ordre : $\{2,15,22,3,14,40,43\}$.

Q 3 . Ecrivez une méthode `main` qui :

1. crée l’arbre binaire ci-dessous :



2. affiche la hauteur et la taille de cet arbre,
3. affiche les valeurs des nœuds dans l’ordre obtenu par un parcours en largeur d’abord,
4. affiche les valeurs des nœuds dans l’ordre obtenu par un parcours en profondeur d’abord.

Q 4 . Arbres binaires ordonnés.

Les arbres binaires ordonnés (ABO dans la suite) sont un cas particuliers d’arbres binaires tels que :

- ▷ il existe une relation d’ordre totale sur les valeurs des nœuds de l’arbre, et donc les valeurs des nœuds sont comparables 2 à 2,
- ▷ les sous-arbres gauche et droit d’un ABO sont des ABO et
  - toutes les valeurs des nœuds du sous-arbre gauche d’un ABO sont inférieures ou égales à la valeur de la racine de l’ABO,
  - toutes les valeurs des nœuds du sous-arbre droit d’un ABO sont supérieures strictement à la valeur de la racine de l’ABO.

  L’arbre vide est un ABO particulier.

La figure 3 présentent des exemples d’ABO (pour des contre-exemples, les 2 arbres de droite dans la figure 1 ne sont pas des ABO).

L’intérêt d’un ABO est qu’il est beaucoup plus rapide de retrouver une valeur dans celui-ci en la comparant avec celle de la racine.

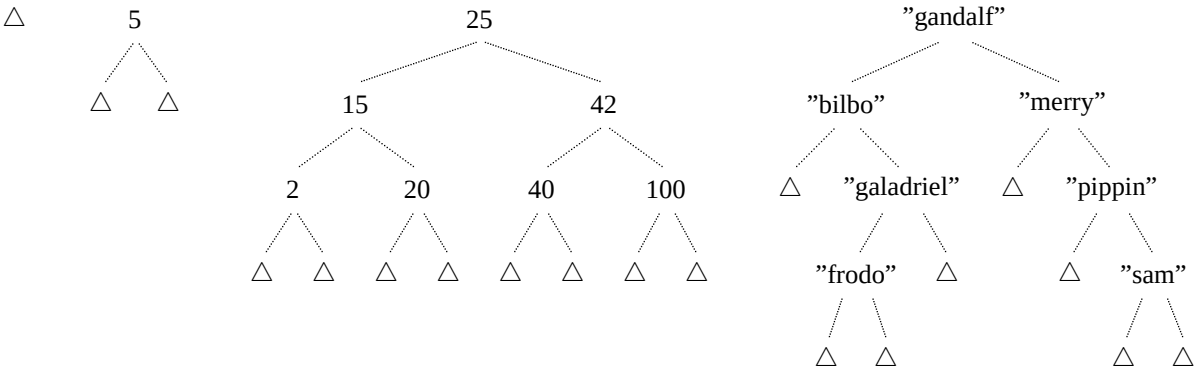


Figure 3: 4 exemples d’arbres binaires, de gauche à droite : l’arbre vide, un arbre réduit, un arbre dont les valeurs sont des entiers et la relation d’ordre est l’ordre naturel sur les entiers et un autre arbre dont les valeurs des nœuds sont des chaînes de caractères et l’ordre est l’ordre lexicographique (càd “alphabétique”).

▷ Donnez le code java d’un classe `ABOrdonne` qui permet de modéliser des arbres binaires ordonnés. Le code de vos méthodes devra tenir compte des propriétés des ABO lorsque cela a un intérêt.

Pour les méthodes dont le code serait identique à celui de la classe `ArbreBinaire` de la question 1, vous pouvez remplacer le corps par le texte :

```
... identique à ArbreBinaire ...
```

Annexe

Interface AB

```
package arbresbin;

/** Type abstrait Arbre Binaire */
public interface AB {
    /** @return le sous-arbre gauche */
    public AB SAGauche() ;
    /** @return le sous-arbre droit */
    public AB SADroit() ;
    /** @return la valeur associée à la racine de l'arbre */
    public Object valeur() ;
    /** @return <t>true</t> si et seulement si cet arbre est une feuille */
    public boolean estFeuille() ;
    /** @return <t>true</t> si et seulement si cet arbre est vide */
    public boolean estVide() ;
    /** @return la hauteur de l'arbre */
    public int hauteur() ;
    /** @return  la taille de l'arbre */
    public int taille() ;
    /** @param v la valeur dont on teste la présence dans l'arbre
      * @return <t>true</t> si et seulement si un noeud de cet arbre
      * contient la valeur <i>v</i>
      */
    public boolean contient(Object v);
    /**  Remplace toutes les valeurs des noeuds égales (au sens
      * equals()) à l'objet <i>val</i> par l'objet <i>nouveau</i>.
      * Le résultat est le nombre de substitutions effectuées.
      * @param val la valeur à remplacer
      * @param nouveau la valeur de remplacement
      * @return le nombre de substitutions effectuées
      */
    public int substitue(Object val, Object nouveau);
}
```

Interface ABVisiteur

```
package arbresbin;

/** Type abstrait Visiteur pour arbres binaires : définit un ordre de parcours
des valeurs des noeuds de l'arbre  */
public interface ABVisiteur{
   /** @return les valeurs contenues dans les noeuds de l'arbre binaire
     *  dans l'ordre de parcours défini par ce visiteur.
     */
   public Iterator lesNoeuds();
}
```