S3 2015-2016

IUT D'ORSAY

Contrôle continu

Durée : 1 heure.

Seuls les transparents de cours sont autorisés.

Exercice 1 Listes récursives

Question 1.1 Écrire une fonction **NbOccurences** qui, étant donné une information (entier) **val** et une liste d'entiers **L** passés en paramètre, renvoie le nombre de fois que l'information **val** donnée apparaît dans **L**.

Correction

1. cas d'arrêt : liste vide : retourner 0
2. cas général : Si on connaît le nombre d'occurences c de **val** dans la liste **uneListe.reste()**, alors on retourne $c + 1$ si **uneListe.tete()** = **val** sinon on retourne c.

Algorithm 1: Fonction **NbOccurences**(uneListe,val) retourne (entier)

```
/* {retourne la somme des valeurs enregistrées dans une liste.} */
paramètres
  (D) uneListe : ListeRéc {liste d'entiers}
  (D) val : Info
begin
  if uneListe.vide() then
    retourner (0);
  else
    if (uneListe.tete()=val) then
      retourner NbOccurences(uneListe.reste(),val) +1;
    else
      retourner NbOccurences(uneListe.reste(),val) ;
    end
  end
end
```

□

Question 1.2 Écrire une fonction **supprimerVal** qui retourne la liste résultant de la suppression d'une information **val** dans une liste passée en paramètre. Si **val** s'y trouve plusieurs fois, toutes les occurrences seront supprimées, si **val** ne s'y trouve pas, la liste retournée est identique à la liste passée en paramètre.

Correction

1. cas d'arrêt : liste vide : retourner la liste de vide
2. cas général : Supposons, qu'on a la liste extraite X de *uneListe.reste*() sans occurence de **val** . Si **uneListe.tete**() = **val**, alors il faut supprimer le premier element. Donc on retourne X.
   Sinon, on retourne la liste X avec l'ajout du premier element de **uneListe**,

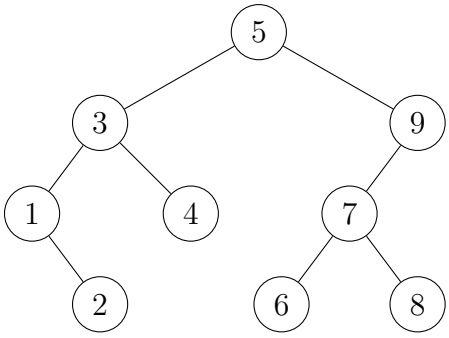
Algorithm 2: Fonction **supprimerVal**(uneListe,val) retourne (ListeRéc)

```
/* {retourne la somme des valeurs enregistrées dans une liste.} */
Paramètres :
  (D) uneListe : ListeRéc {liste d'entiers}
  (D) val : Info
Variable :
  (D) L1 : ListeRéc {liste d'entiers}
begin
  if uneListe.vide() then
    retourner (uneListe);
  else
    if uneListe.tete() != val then
      retourner supprimerVal(uneListe.reste(),val) ;
    else
      L1 = supprimerVal(uneListe.reste(),val) ;
      retourner L1.préfixer(uneListe.tete()) ;
    end
  end
end
```

□

Exercice 2 Arbres binaires

Soit l'arbre binaire (de recherche) suivant :



Question 2.1 Exprimer le père, le ou les fils ainsi que la profondeur du sommet 3. Donner tous les sommets intérieurs et les feuilles de l'arbre. Enfin, donner la taille et la hauteur de l'arbre.

Correction

Voir le cours.

□

Exercice 3 Encore des arbres binaires.

Soit deux arbres binaires $\mathbf{A_1}$ et $\mathbf{A_2}$. On dit que l'arbre $\mathbf{A_1}$ est *contenu* dans l'arbre $\mathbf{A_2}$ si, toutes les valeurs présentes dans $\mathbf{A_1}$ sont aussi présentes dans $\mathbf{A_2}$. Nous supposerons que nous disposons d'une fonction **contientValeur**$(\mathbf{val}, \mathbf{A})$ qui retourne vrai si la valeur **val** est présente dans l'arbre **A**.

L'objectif de cet exercice est d'écrire une fonction **contenuArbre**$(\mathbf{A_1}, \mathbf{A_2})$ qui détermine si $\mathbf{A_1}$ est contenu dans $\mathbf{A_2}$.

Question 3.1 Donner la valeur que retournerait la fonction **contenuArbre**$(\mathbf{A_1}, \mathbf{A_2})$

1. si $\mathbf{A_1}$ et $\mathbf{A_2}$ sont des arbres vides.
2. si $\mathbf{A_1}$ est l'arbre vide et si $\mathbf{A_2}$ n'est pas l'arbre vide.
3. si $\mathbf{A_1}$ n'est pas l'arbre vide et si $\mathbf{A_2}$ est l'arbre vide.

Correction

1. Si $\mathbf{A_1}$ et $\mathbf{A_2}$ sont des arbres vides, alors il faut retourner vrai.
2. Si $\mathbf{A_1}$ est l'arbre vide et si $\mathbf{A_2}$ n'est pas l'arbre vide, alors il Il faut retourner vrai.
3. Si $\mathbf{A_1}$ n'est pas l'arbre vide et si $\mathbf{A_2}$ est l'arbre vide, alors il faut retourner faux.

□

Question 3.2 Écrire la fonction **contenuArbre**$(\mathbf{A_1}, \mathbf{A_2})$.

Correction

1. cas d'arrêt :
   (a) voir la question précédente
   (b) si $\mathbf{A_1}.info()$ n'est pas contenu dans $\mathbf{A_2}$), alors il faut retourner faux
2. cas général :
   Il faut vérifier que chaque sous-arbre de $\mathbf{A_1}$ soit contenu dans dans $\mathbf{A_2}$.

Algorithm 3: Fonction **contenuArbre**($\mathbf{A_1}, \mathbf{A_2}$) retourne (booléen)

```
/* {retourne vrai si l'arbre A1 est contenu dans A2.}          */
Paramètres :
  (D) (A1, A2) : deux arbres binaires contenant des entiers
Variable :
  (D) b : booléen
begin
  if A1.vide() then
    retourner vrai;
  else
    b = contientValeur(A1.info,A2) ;
    if b = vrai then
      retourner contenuArbre(A1.filsgauche(), A2) ET
                          contenuArbre(A1.filsdroit(), A2) ;
    else
      retourner b ;
    end
  end
end
```

□

Exercice 4 Arbres binaires de recherche

L'objectif de cet exercice est d'écrire une fonction **CompterValeurSup** qui retourne le nombre d'information ayant une valeur **strictement** supérieure à **val** dans l'arbre **A** passé en paramètre. Nous supposerons que l'arbre ne contient pas deux valeurs identiques.

Question 4.1 Donner la valeur que retournerait la fonction **CompterValeurSup**(**val**,**A**).

1. si **val** = 0 et si **A** est l'arbre de la figure de l'exercice 2.
2. si **val** = 2 et si **A** est l'arbre de la figure de l'exercice 2.
3. si **val** = 3 et si **A** est l'arbre de la figure de l'exercice 2.

Correction

1. si **val** = 0 et si **A** est l'arbre de la figure de l'exercice 2.

   9

2. si **val** = 2 et si **A** est l'arbre de la figure de l'exercice 2.

   7

3. si **val** = 3 et si **A** est l'arbre de la figure de l'exercice 2.

   6

□

Question 4.2 Écrire la fonction **CompterValeurSup**

Correction

1. cas d'arrêt :
   (a) si arbre est vide : retourner 0
   (b) si l'information contenu par la racine est plus petite que **val** : retourner 0
2. cas général :
   (a) si l'information contenue par la racine est égal que **val** : alors il faut simplement compter le nombre de sommet de l'arbre fils droit de A (car il est un arbre de recherche).
   (b) si l'information contenue par la racine est $>$ que **val** : alors il faut tout d'abord compter le nombre de sommet de l'arbre fils droit de A (car il est un arbre de recherche). Il faut aussi compter le nombre de sommets ayant une information plus supérieure que **val**. Pour l'arbre gauche, certains sommets (mais pas tous) ont ayant une information inférieure à **val**.

Algorithm 4: Fonction **CompterValeurSup**(**val**,**A**) retourne (entier)

```
/* {retourne le nombre d'élément supperieur à val dans A.}      */
Paramètres :
  (D) A : un arbre binaire de recherche
  (D) val : Info
Variable :
  (D) b : booléen
begin
  if A.vide() then
    retourner 0;
  else
    b = contientValeur(A1.info,A2) ;
    if A.info < val then
      retourner 0;
    else
      if A.info == val then
        retourner CompterValeurSup(val,A.filsdroit());
      else
        retourner CompterValeurSup(val,A.filsdroit())
                          + CompterValeurSup(val,A.filsgauche());
      end
    end
  end
end
```

□