S3 2015-2016

UNIVERSITÉ PARIS SUD
IUT D'ORSAY

Contrôle continu

Exercice 1 Listes récursives

Question 1.1 Écrire une fonction **Nbelements** qui, étant donné une liste d'entiers **L** passé en paramètre, renvoie le nombre d'éléments de la liste d'entiers **L**.

1. Donner le ou les cas d'arrêt
2. Expliquer le cas général.
3. Écrire la fonction **Nbelements**(**L**)

Correction

1. cas d'arrêt : liste vide : retourner 0
2. cas général : Si on connaît le nombre d'éléments c dans la liste **uneListe.reste**(), alors on retourne $c + 1$.

Algorithm 1: Fonction **RangListe**(**uneListe**) retourne (entier)

```
/* {retourne le nombre d'éléments de la liste d'entiers.}      */
paramètres
  (D) uneListe : ListeRéc {liste d'entiers}
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
```

□

4. Dérouler votre fonction **Nbelement** sachant que **L**=< 2, 2, 4, 4 >.

Question 1.2 Écrire une fonction **appartenance** qui renvoie vrai si une information **val** donnée se trouve dans une liste ordonnée (par ordre croissant) **L** passée en paramètre et faux sinon.

1. Donner le ou les cas d'arrêt
2. Expliquer le cas général.
3. Écrire la fonction **appartenance**(**val**, **L**)

Correction

1. cas d'arrêt :
   (a) si liste vide : retourner $faux$
   (b) si val est égal à la valeur contenu par le premier élément retourner $vrai$
   (c) si val est $>$ à la valeur contenu par le premier élément retourner $faux$
2. cas général : On parcours la liste jusqu'à tomber sur un cas d'arrêt.

Algorithm 2: Fonction **appartenance**(**val**, **L**) retourne (booléen)

```
/* {retourne le nombre d'éléments de la liste d'entiers.}      */
paramètres
  (D) L : ListeRéc {liste d'entiers}
  (D) valeur : info {un entier}
begin
  if uneListe.vide() then
    retourner (faux);
  else
    if (uneListe.tete()=val) then
      retourner (vrai);
    else
      if (uneListe.tete()>val) then
        retourner (faux);
      else
        retourner appartenance(val, L.reste()) ;
      end
    end
  end
end
```

□

4. Dérouler votre fonction **appartenance(val,L)** sachant que **L**$=< 2, 2, 4, 4, 7 >$ et **val**=6.

Question 1.3 Écrire une fonction **NbSup** qui, étant donné une liste d'entiers **L** d'entiers triée par ordre croissant, et un entier **val** passés en paramètre, renvoie le nombre d'information enregistrée la liste **L inférieures ou égales** à **val**. Nous supposerons que la liste ne peut contenir des valeurs identiques.

Si **L** $=< 2, 2, 4, 4, 5, 6, 7 >$, la fonction **NbSup**(**L**,4) devra retourner 4.

1. Donner le ou les cas d'arrêt
2. Expliquer le cas général.

3. Écrire la fonction **NbSup**(**L**, **val**).

Correction

1. cas d'arrêt : liste vide : retourner 0
2. si val est $>$ à la valeur contenu par le premier élément retourner 0
3. cas général : Supposons, qu'on a x éléments plus inférieur à **val** dans la liste *uneListe.reste*(). Si **uneListe.tete**() $<$ **val**, alors alors on incrémente x

Algorithm 3: Fonction **NbSup**(uneListe,val) retourne (ListeRéc)

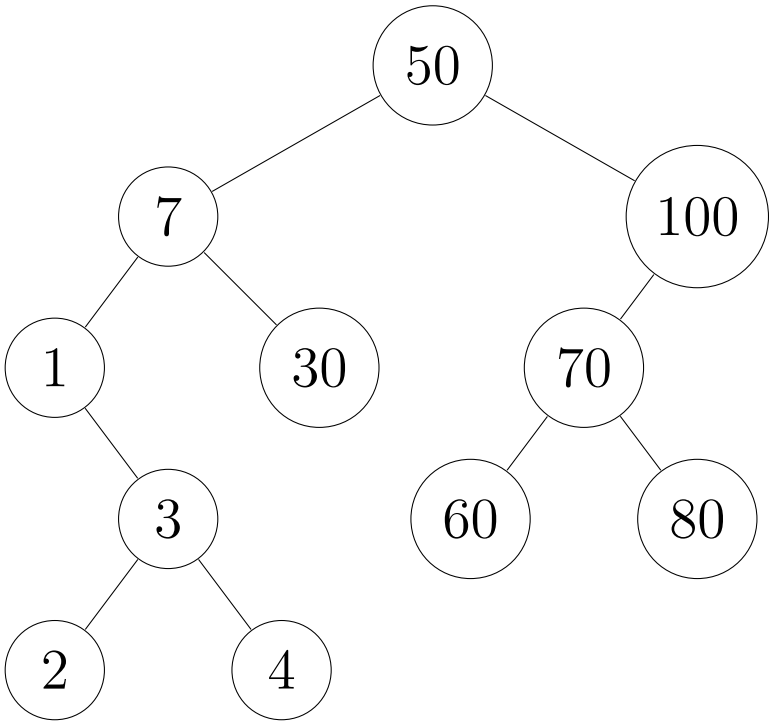
```
/* {retourne la somme des valeurs enregistrées dans une liste.} */
Paramètres :
  (D) uneListe : ListeRéc {liste d'entiers}
  (D) val : Info
Variable :
  (D) L1 : ListeRéc {liste d'entiers}
begin
  if uneListe.vide() then
    retourner (0);
  else
    if uneListe.tete<val then
      retourner 0 ;
    else
      retourner NbSup(uneListe.reste(),val) +1 ;
    end
  end
end
```

□

4. Dérouler votre fonction **NbSup** sachant que **L**=$< 2, 2, 4, 4, 5, 6, 7 >$ et **val**=4.

Exercice 2 Arbres binaires

Soit l'arbre binaire suivant avec 50 comme racine :



Question 2.1 Exprimer

1. le père, le ou les fils du sommet 50
2. le père, le ou les fils du sommet 7

Question 2.2 Donner tous les feuilles de l'arbre.

Question 2.3 Donner la taille et la hauteur de l'arbre.

Correction

Voir le cours.

□

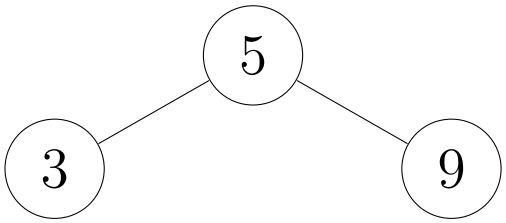
Question 2.4 Est-il un arbre binaire de recherche ? Justifier votre réponse.

Exercice 3 Arbres binaires de recherche

L'objectif de cet exercice est d'afficher les valeurs contenues par les sommets de l'arbre binaire de recherche de façon croissante. Nous utiliserons la procédure **Ecrire(info)** qui affiche l'information **info**.

Question 3.1 Donner l'affichage que retournerait la procédure **afficheOrdre**(**unArbre**).

1. si **unArbre** est vide ;
2. si **unArbre** est feuille ;
3. si **unArbre** est l'arbre de la figure dessous :



Correction

1. si **unArbre** est vide : la procédure n'affiche rien
2. si **unArbre** est feuille : la procédure affiche l'info enregistrée dans la feuille
3. si **unArbre** est l'arbre de la figure dessous : la procédure affiche < 3 5 9 >

□

Question 3.2 Écrire la procédure **afficheOrdre**(**unArbre**).

Correction

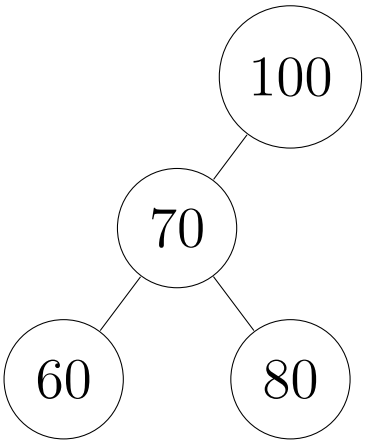
1. cas d'arrêt : voir la question précédente
2. cas général :
   (a) Comme toute l'information contenu dans de l'arbre fils gauche est inférieur à **unArbre**.info(), il faut d'abord afficher toutes les informations de l'arbre fils gauche, puis **unArbre**.info()
   (b) Ensuite, il faut afficher toutes les informations de l'arbre fils droit.

Algorithm 4: Procédure **afficheOrdre**(**unArbre**)

```
/* {Afficher les valeurs contenues par les sommets de l'arbre
   binaire de recherche de façon croissante}                    */
Paramètres :
  (D) unArbre : un arbre binaire de recherche
begin
  if unArbre.vide() then
    Ne rien faire;
  else
    afficheOrdre(unArbre.filsgauche() ;
    Ecrire (unArbre.info()) ;
    afficheOrdre(unArbre.filsdroit() ;
  end
end
```

□

Question 3.3 Dérouler votre fonction sur l'exercice de la figure suivante :



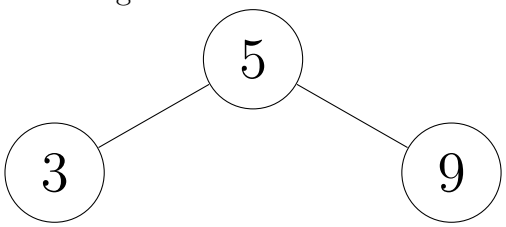
Exercice 4 Encore des arbres binaires de recherche

L'objectif de cet exercice est d'écrire une fonction **SommeElement** qui, étant donné un arbre binaire de recherche **A** et une valeur **val** passé en paramètre, retourne la somme des valeurs contenues dans l'arbre ayant une valeur inférieure à **val**. Nous supposerons que l'arbre ne contient pas deux valeurs identiques.

Question 4.1 Donner la valeur que retournerait la fonction

SommeElement(**unArbre**,**val**).

1. si **unArbre** est vide ;
2. si **unArbre** est feuille ;
3. si **unArbre** est l'arbre de la figure dessous et **val** = 5.
4. si **unArbre** est l'arbre de la figure dessous et **val** = 1.



Question 4.2 Écrire une fonction **SommeElement**(**unArbre**,**val**)

Correction

1. cas d'arrêt : voir la question précédente
2. cas général :
   (a) Comme toute l'information contenu dans de l'arbre fils gauche est inférieur à **unArbre**.info(), il faut d'abord afficher toutes les informations de l'arbre fils gauche, puis **unArbre**.info()
   (b) Ensuite, il faut afficher toutes les informations de l'arbre fils droit.

Algorithm 5: fonction **SommeElement**(**unArbre**,**val**) retourne un entier

```
/* {}                                                              */
Paramètres :
  (D) unArbre : un arbre binaire de recherche
  (D) val : un entier
variable locale :
  (D) x : un entier
begin
  if unArbre.vide() then
    retourner 0;
  else
    if unArbre.info() > val then
      retourner 0;
    else
      x ←SommeElement(unArbre.filsgauche(),val) ;
      x ← x+SommeElement(unArbre.filsdroit(),val) ;
      retourner (unArbre.info()) +x ;
    end
  end
end
```

□