Université de Nice-Sophia Antipolis — POLYTECH
PeiP2 — 2015–2016

Examen de Algorithmique/Programmation

Durée : 2h
Aucun document autorisé
Mobiles interdits *Note : la qualité des commentaires, avec notamment la présence d'affirmations significatives, ainsi que les noms donnés aux variables, l'emploi à bon escient des majuscules et la bonne indentation rentreront pour une part importante dans l'appréciation du travail.*

I Récursivité et listes

On définit récursivement l'ensemble des listes $\mathcal{L}$ par

$$\mathcal{L} = \emptyset \mid \mathcal{E} \times \mathcal{L}$$

avec l'ensemble $\mathcal{E}$ des éléments de la liste, et $\emptyset$ l'ensemble vide (absence de liste). Les fonctions suivantes permettent de manipuler une liste.

estvide	:	$\mathcal{L}$	$\rightarrow$ *booléen*
tête	:	$\mathcal{L}$	$\rightarrow \mathcal{E}$
queue	:	$\mathcal{L}$	$\rightarrow \mathcal{L}$
cons	:	$\mathcal{E} \times \mathcal{L}$	$\rightarrow \mathcal{L}$

La fonction *estvide* renvoie *vrai* si la liste est vide et *faux* sinon. La fonction *tête* renvoie le premier élément (s'il existe) de la liste et la fonction *queue* renvoie la liste moins l'élément de tête. Enfin, fonction *cons* crée une liste par concaténation d'un élément et d'une liste.

Note : pour les questions 2, 3 et 4, vous utiliserez ces 4 fonctions

► 1. Donnez les axiomes (les équations) qui expriment de façon formelle la sémantique des quatre fonctions précédentes (et au pire, si vous ne savez pas donner les axiomes, expliquez de manière informelle cette sémantique).

Question sur 1 pt

1. estvide(listevide) = vrai
2. $\forall x \in \mathcal{E}, \forall l \in \mathcal{L}$, esvide(cons(x,l)) = faux
3. $\forall x \in \mathcal{E}, \forall l \in \mathcal{L}$, tête(cons(x,l)) = x ,
4. $\nexists$ l, tête(listevide)=l
5. $\forall x \in \mathcal{E}, \forall l \in \mathcal{L}$, queue(cons(x,l)) = l
6. $\nexists$ l, queue(listevide)=l
7. $\forall l \in \mathcal{L}$,cons(tête(l), queue(l)) = l

. . . .

► 2. Écrivez récursivement et de façon algorithmique la fonction *longueur(l : liste)* qui renvoie la longueur (son nombre d'éléments $\geqslant 0$) de la liste l passée en paramètre.

Question sur 2 pts

```
{Rôle : renvoie le nombre d'éléments de la liste l }
fonction longueur(l : liste) : naturel
début
   si estvide(l) alors rendre 0
      sinon
          rendre longueur(queue(l))+1
   finsi
finfonc
```

. . . .

► 3. Écrivez de façon algorithmique deux versions, une itérative et une récursive, de la fonction *ième(l : liste, n : entier)* qui renvoie l'élément de rang n dans la liste l. Vous vérifierez que le rang est valide, *i.e.* $1 \leqslant n \leqslant$*longueur(l)*. En cas d'erreur, vous appellerez simplement la fonction `erreur`.

Question sur 3 pts

version itérative :

```
{Antécédent :  1 ⩽ n ⩽ longueur(l) }
{Rôle : renvoie l'élément de rang n dans la liste l }
fonction ième(l : liste, n : entier) : élément
début
   si  n<1 ou n>longueur(l) alors erreur(rang invalide)
   sinon
       pourtout i de 1 à n-1 faire
            l ← queue(l)
       finpour
       rendre tête(l)
   finsi
finfonc
```

version récursive :

```
{Antécédent :  1 ⩽ n ⩽ longueur(l) }
{Rôle : renvoie l'élément de rang n dans la liste l }
fonction ième(l : liste, n : entier) : élément
débu
   si  n<1 ou n>longueur(l) alors erreur(rang invalide)
      rendre ième-aux(l,n)
   finsi
finfonc
```

```
fonction ième-aux(l : liste, n : entier) : élément
début
   si n=1 alors rendre tête(l)
      sinon
          rendre ième-aux(queue(l), n-1)
   finsi
finfonc
```

....

▶ 4. Écrivez récursivement et de façon algorithmique la fonction *insérer(l : liste, n : entier, x : élément)* qui insère dans une liste l au rang n l'élément x. La fonction renvoie la liste l dans laquelle x a été inséré. Comme précédemment, vous vérifierez que le rang est valide.

Question sur 3 pts

```
{Antécédent :  1 ⩽ n ⩽ longueur(l)+1 }
{Rôle : renvoie la liste l dans laquelle l'élément x
        a été inséré au rang n
}
fonction insérer(l: liste, n: entier, x: élément): liste
début
      si  n<1 ou n>longueur(l)+1 alors erreur(rang invalide)
        sinon rendre insérer-aux(l, n, x)
      finsi
finfonc

fonction insérer-aux(l: liste, n: entier, x: élément): liste
début
     si n=1 alors rendre cons(x,l)
       sinon
           rendre cons(tête(l), insérer-aux(queue(l),n-1,x))
     finsi
finfonc
```

....

II Tri

▶ 5. Écrivez de façon algorithmique ou en JAVA, le tri par insertion séquentielle. N'oubliez pas de bien spécifier l'invariant.

Question sur 4 pts

Version algoritmique :

```
{Rôle : trie la liste l en ordre croissant des clés}
procédure triInsertSeq(l : liste)
début
   pourtout i de 2 à longueur(l) faire
      {Inv : la sous-liste de ième(l,1) à ième(l,i-1) est triée}
      x ← ième(l,i)
      j ← i-1
      sup ← vrai
      tantque j>0 et sup faire
         {on décale et on cherche le rang d'insertion}
         {simultanément de façon séquentielle}
         si clé(ième(l,j))⩽clé(x) alors sup ← faux
            sinon {on décale}
                  ième(l,j+1) ← ième(l,j)
                  j ← j-1
         finsi
      fintantque
      ième(l,j+1) ← x
   finpour
finproc
```

Version Java :

```
/*
 *  Rôle : tri la liste courante selon la méthode d'insertion séquentielle
 */
public void  triInsertSeq() {
    for (int i=1; i<l.size(); i++) {
        Élément<C,V> e = l.get(i);
        //la liste get(1)..get(i-1) est triée
        int j = i-1;
        //rechercher le rang d'insertion
        while (j>=0 && e.clé().compareTo(l.get(j).clé())<0) {
            //on décale et on cherche le rang
            //d'insertion simultanément de
            //façon séquentielle
            l.set(j+1,l.get(j));
            j--;
        }
        //j+1 est ce rang d'insertion
        if (j+1!=i) l.set(j+1,e);
    }
}
```

....

▶ 6. Donnez le nombre exact de comparaisons et de transferts pour une liste de longueur n dans le pire des cas, le cas moyen et dans le meilleur des cas. Vous avez donc 6 valeurs à donner.

Question sur 2 pts :

	Nb comparaisons	Nb Transferts
le pire	$\frac{1}{2}(n^2-n)=\mathcal{O}(n^2)$	$\frac{1}{2}(n^2+3n-4)=\mathcal{O}(n^2)$
moyen	$\frac{1}{4}(n^2-n)=\mathcal{O}(n^2)$	$\frac{1}{4}(n^2+7n-8)=\mathcal{O}(n^2)$
meilleur	$n-1=\mathcal{O}(n)$	$2n-2=\mathcal{O}(n)$

....

III Arbres binaires

Un arbre binaire implémente l'interface suivante :

```
interface ArbreBinaire<T> {
   boolean estVide();
   ArbreBinaire sag() throws ArbreVideException;
   ArbreBinaire sad() throws ArbreVideException;
   T racine() throws ArbreVideException;
}
```

► 7. Écrivez en Java la méthode récursive `nbNoeuds` qui, renvoie le nombre nœuds (incluant les feuilles) de l'arbre courant.

Question sur 2 pts

```
/**
 * Rôle : renvoie le nombre de noeuds de l'arbre courant
 */
public int nbNoeuds() throws ArbreVideException {
    return estVide() ? 0 :
                  1 + sag().nbNoeuds() + sad().nbNoeuds();
}
```

....

► 8. Écrivez en Java la méthode booléenne récursive `estEquilibré` qui renvoie *vrai* si l'arbre binaire courant est équilibré et *faux* sinon. On considérera qu'un arbre est équilibré, si son sous-arbre gauche possède autant de noeuds son sous-arbre droit.

Question sur 2 pts

```
public boolean estEquilibré() throws ArbreVideException {
    return estVide() ? true :
                     sag().nbNoeuds() == sad().nbNoeuds();
}
```

....

► 9. Écrivez en Java une méthode récursive `numéroter`, qui numérote tous les nœuds de l'arbre courant en partant de 1, selon l'ordre GND, comme le montre l'exemple ci-dessous :

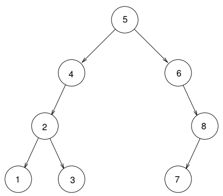

Figure 1 – Exemple d'arbre binaire numéroté

Vous pourrez ajouter une méthode `setRang`, qui donne à l'attribut `valeur` le numéro de rang du nœud.

Question sur 3 pts

```
public void numéroter() throws ArbreVideException {
    if (estVide()) throw new ArbreVideException();
    numéroter(this, 1);
}

private int numéroter(ArbreBinaire<T> a, int n)
throws ArbreVideException
{
    if (a.estVide()) return n;
    n = numéroter(a.sag(), n);
    a.setRang(n++);
    return numéroter(a.sad(), n);
}
```

....