POLYTECH – PeiP2 2016–2017

Contrôle de Algorithmique et Programmation JAVA

Durée : 1h30 Aucun document autorisé

Nom : Prénom : Groupe :

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.

1 Recherche

▶ 1. Écrivez de façon algorithmique et récursive la fonction `rechercheDichoR` qui recherche un élément `E` de clé `c` de façon dichotomique dans une liste linéaire ordonnée `lst`. Si la clé n'est pas trouvée, il faudra évidement le signaler. Vous donnerez la complexité de cette recherche dans le pire des cas. Rappel :

longueur : $\mathcal{L}_o \rightarrow naturel$
ième : $\mathcal{L}_o \times naturel \rightarrow \mathcal{E}$

L'en-tête de cette fonction est le suivant :

```
{ Antécédent : à compléter
  Rôle : ....
}
fonction rechercheDichoR(lst, c, gauche, droite) : E
```

2 Liste ordonnée

▶ 2. On définit récursivement l'ensemble des listes ordonnées $\mathcal{L}_o$ comme : $\mathcal{L}_o = \emptyset \mid \mathcal{E} \times \mathcal{L}_o$, avec $\mathcal{E}$ l'ensemble des éléments (valeur + clé) de la liste muni d'une relation d'ordre sur la clé. Donnez les axiomes de l'opération supprimer qui supprime un élément de clé c dans une liste ordonnée l. On considérera la relation d'ordre « inférieur ou égal ». La signature de l'opération supprimer est la suivante :

supprimer : $\mathcal{L}_o \times \mathcal{C}lé \rightarrow \mathcal{L}_o$

3 Tri

Dans cet exercice, on se propose d'étudier un algorithme de tri appelé tri à bulles (*bubble sort*). Le tri à bulles permute les éléments mal ordonnés, en faisant « *remonter* » les éléments les plus petits vers le début de la liste. On présente ci-dessous une version algorithmique et itérative de cette méthode de tri.

```
fonction triBulles(lst : liste d'éléments comparables)
   faire
        échangés ← faux
        pourtout i de longueur(lst)-1 à 1 faire
            si clé(ième(lst, i+1)) < clé(ième(lst, i)) alors
               échanger(ième(lst, i+1), ième(lst, i))
               échangés ← vrai
            finsi
        finpour
   tantque échangés
finfonc
```

▶ 3. Proposez une implémentation en Java.

► 4. Quelle est la complexité (en nombre de comparaisons effectuées) de cet algorithme sur une liste de longueur n si :
- la liste est déjà triée dans l'ordre croissant ;
- la liste est triée dans l'ordre décroissant.

► 5. En admettant que sa complexité moyenne soit similaire à sa complexité dans le pire des cas, comment le tri à bulles se compare-t-il au tri par insertion et au tri rapide, dont vous rappellerez les complexités moyennes (toujours en nombre de comparaisons effectuées) ?

▶ 6. À la fin de la k-ème itération de la boucle **tantque**, quels sont les éléments bien triés à coup sûr ? Utilisez cette observation pour proposer une amélioration de l'algorithme. À votre avis, dans quelle mesure la complexité (en nombre de comparaisons) de la méthode est-elle modifiée ?

4 Arbre Binaire

On rappelle qu'un arbre binaire implémente l'interface suivante :

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

▶ 7. Écrivez en Java une méthode `échangerSousArbre` qui échange le sous-arbre gauche et le sous-arbre droit de l'arbre courant.

▶ 8. En TD, nous avons développé une méthode qui calcule la hauteur d'un arbre binaire. En utilisant cette méthode sans la récrire, écrivez en Java la méthode `trierSaParHauteur` qui transforme l'arbre courant de telle sorte que pour tout noeud de cet arbre, la hauteur du sous-arbre gauche soit plus petite que celle du sous-arbre droit (c'est-à-dire que s'il existe un noeud de l'arbre qui ne vérifie pas cette propriété, la fonction doit inverser les deux sous-arbres de ce noeud). Remarque : ne pas gérer l'exception `ArbreVideException` dans un **`try-catch`**.