Licence **Sciences et Technologies**	**ANNEE : 2007/2008** **SESSION D'AUTOMNE 2007** **ETAPE :** CSB3, MHT53 **UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** 17 Décembre 2007 **Heure :** 8 H 30 **Durée :** 3 H **Documents :** Tous documents interdits **Vous devez répondre directement sur le sujet qui comporte 10 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs** **Epreuve de** M^{me} Delest.

Indiquez votre code **d'anonymat** : N° : .

La notation tiendra compte de la clarté de l'écriture des réponses.

Barème indicatif

- Question 1 – Connaissances générales : 4 points
- Question 2 – Tas : 1.5 points
- Question 3 – Arbre AVL : 1.5 points
- Question 4 – Utilisation des structures de données : 4 points
- Question 5 – Ecriture de fonction sur les arbres : 2 Points
- Question 6 – Ecriture de fonction sur les piles : 2 Points
- Question 7 – Listes et arbres binaires: 6 points

Question 1. Cochez les affirmations qui sont correctes :

□ Le temps d'accès au dernier élément d'une liste simplement chainée est en O(n).

□ Dans un arbre binaire de recherche le minimum est toujours à la racine de l'arbre.

□ Le temps d'accès à l'élément maximum d'un tas min est en O(1).

□ Un arbre AVL est un tas.

□ Pour un arbre binaire de recherche donné l'obtention de la liste des nombres triés est en temps O(n).

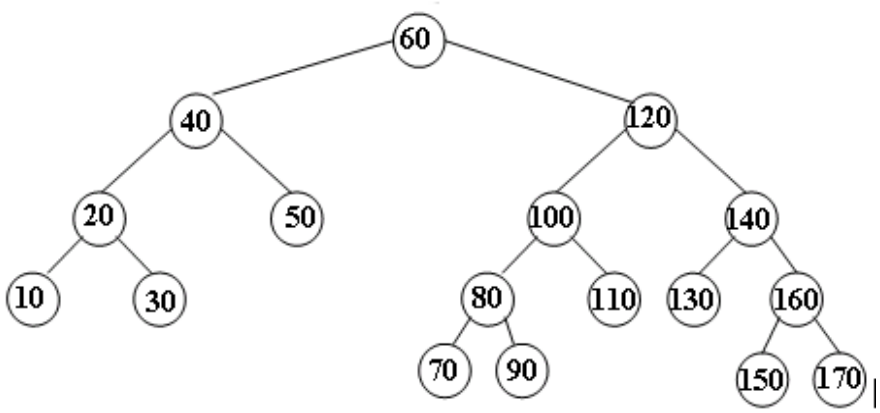
□ Dans un tas, la primitive supprimerValeur consiste à supprimer une valeur située dans une feuille.

□ La complexité mémoire d'une table de hachage chainée est plus importante que la complexité mémoire d'une table de hachage ouvert

□ Si s est une variable de type ^entier, on modifie la valeur de l'entier en écrivant s=s+1

Question 2. Soit la suite de clé 5,8,2,6,3,4,1,7. Construire le tas-Min correspondant à l'insertion consécutive des clés, on dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final. Montrez l'exécution de *supprimerValeur* sur le tas ainsi construit.

Question 3. On donne l'arbre AVL suivant ;



1 – Donnez sur l'arbre ci-dessus les positions des insertions de feuille qui conduiront à un arbre déséquilibré ?

2 - Donnez sur le dessin les facteurs d'équilibrage.

3 - Dessinez et expliquez les modifications de l'arbre lors de l'insertion de la valeur 65. On mentionnera les modifications des facteurs d'équilibrages.

Question 4. Un dispositif dispose de deux robots pour traiter des commandes. Une commande est identifiée par un nom sur 30 caractères et une date sous la forme [jour,mois,année]. Le premier robot (R1) mémorise les commandes dès quelles lui parviennent. Le second robot (R2) traite les commandes.
1 – Quelle structure de données pertinente peut-on utiliser pour accéder en temps constant aux identificateurs des commandes ?
2 – Quelle structure de données peut utiliser R1 pour que R2 traite les commandes dans le même ordre où elles arrivent à R1 ?
3 – Décrivez le type de ces structures et notamment on détaillera les éléments de chaque structure.
4 – Ecrire les fonctions *ajouterCommande* (utilisée par R1) et *supprimerCommande* utilisé par R2.
5 – Modifier la fonction *ajouterCommande* pour éviter les doublons.

Question 5. On définit la longueur de cheminement externe d'un arbre binaire comme la somme des hauteurs des feuilles (nombre d'arêtes de la racine à la feuille). Ecrire la fonction longueurDeCheminement qui prend en paramètre un arbre et calcule ce paramètre en utilisant uniquement les primitives du type abstrait.

Question 6. Ecrire une fonction qui prend en entrée une pile d'entier et qui fournit en sortie la pile modifiée telle que tous les nombres multiples de 3 se retrouvent en tête de pile et dans le même ordre que la pile. Par exemple, [1,3,5,4,2,6,8 est transformée en [1,5,4,2,8,3,6.

Question 7. On considère un tableau de dimension N contenant des entiers tous différents appartenant à l'intervalle [1..N]. Par exemple, le tableau de dimension 8 contient 5,8,2,6,3,4,1,7.

1 – Ecrire une fonction *tableauToListe* qui transforme un tableau en une liste simplement chainée en conservant l'ordre des entiers dans le tableau et renvoie vrai si les éléments du tableau sont conforme à l'énoncé (des entiers tous différents) et faux sinon.

2 – Ecrire une fonction *picDeListeSC* qui à partir de la liste ainsi constituée fournit la liste des valeurs T[j] telles que T[j-1]<T[j] et T[j]>T[j+1] (sur l'exemple (4,6,8)).

3 – Si on choisit d'utiliser des listes doublement chainées, écrire la fonction *picDeListeDC* en utilisant la primitive *précédent*.

4 – Donnez les avantages et les inconvénients des deux implémentations et des deux fonctions :

- En temps
- En mémoire
- En lisibilité de la fonction.

5 – Ecrire une fonction *tableauToABR* qui transforme un tableau en un arbre binaire de recherche renvoie vrai si tous les éléments du tableau sont conformes à l'énoncé (des entiers tous différents) et faux sinon. On donnera l'algorithme COMPLET d'insertion.

6 – Appliquer cette fonction au tableau donné en exemple. Donnez la suite des valeurs dans l'ordre préfixe (tableau PR), dans l'ordre infixe (tableau IN) et dans l'ordre suffixe (tableau SU).

7 - Donner sous formes de schéma ou de tableau la représentation des structures de données dans le cas où l'arbre est implémenté en allocation statique

8 - Donner sous formes de schéma ou de tableau la représentation des structures de données dans le cas où l'arbre est implémenté en allocation dynamique

9 – Ecrire en utilisant les primitives du type abstrait *arbreBinaire* la fonction *compte1NoeudTA* dont le résultat est le nombre de nœud ayant un seul fils.

10 – Ecrire la fonction *compte1NoeudAD* en utilisant directement l'implémentation en allocation dynamique.

Récapitulatif des types et des primitives

Listes simplement chainées (listeSC)

fonction premier(val L:type_liste):^type_predefini;
fonction suivant(val L:type_liste; val P:^type_predefini):^type_predefini;
fonction listeVide(val L:type_liste):booléen;
fonction créer liste(ref L:type_liste):vide;
fonction insérerAprès(val x:type_prédéfini;ref L:type_liste; val P:^type_predefini):vide;
fonction insérerEnTete(val x:type_prédéfini;ref L:type_liste):vide;
fonction supprimerAprès(ref L:type_liste;val P:^type_predefini):vide;
fonction supprimerEnTete(ref L:type_liste):vide;

Listes doublement chainées (listeDC), On ajoute les primitives suivantes

fonction dernier(val L:type_liste):^type_predefini;
fonction précédent(val L:type_liste; val P:^type_predefini):^type_predefini;

Piles

fonction valeur(ref P:pile de type_predefini):type_predefini;
fonction pileVide(ref P:pile de type_predefini):booléen;
fonction empiler(ref P:pile de type_predefini; val v:type_predefini):vide;
fonction dépiler (ref P:pile de type_predefini):vide;
fonction créerPile(P:pile de type_predefini);

Files

fonction valeur(ref F:file de type_predefini):type_predefini;
fonction fileVide(ref F:file de type_predefini):booléen;
fonction enfiler(ref F:file de type_predefini; val v:type_predefini):vide;
fonction défiler (ref F:file de type_predefini):vide;
fonction créerFile(F:file de type_predefini);

ArbresBinaire

fonction valeurSommet(val A:arbreBinaire de type_prédéfini, val S:sommet):valeur_prédéfini;
fonction racine(val A:arbreBinaire de type_prédéfini):sommet;
fonction filsGauche(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction filsDroit(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction père(val A:arbreBinaire de type_prédéfini, val S:sommet):sommet;
fonction créerArbreBinaire(ref A:arbreBinaire de type_prédéfini, val racine:type_prédéfini):vide;
fonction ajouterFilsGauche(val x:type_prédéfini, ref A:arbreBinaire de type_prédéfini,
val S:sommet):vide;
fonction ajouterFilsDroit(val x:type_prédéfini, ref A:arbreBinaire de type_prédéfini,
val S:sommet):vide;
fonction supprimerFilsGauche(ref A:arbreBinaire de type_prédéfini, val S:sommet):vide;
fonction supprimerFilsDroit(ref A:arbreBinaire de type_prédéfini, val S:sommet):vide;

Arbres Binaires en allocation dynamique

fonction valeurSommet(val S:^sommet):valeur_prédéfini;
fonction racine(val A:arbreBinaire de type_prédéfini):^sommet;
fonction filsGauche(val S:^sommet):^sommet;
fonction filsDroit(val S:^sommet):^sommet;
fonction créerArbreBinaire(val racine:type_prédéfini):^sommet;
fonction ajouterFilsGauche(val x:type_prédéfini, val S:^sommet):vide;
fonction ajouterFilsDroit(val x:type_prédéfini, val S:^sommet):vide;
fonction supprimerFilsGauche(val S:^sommet):vide;
fonction supprimerFilsDroit(val S:^sommet):vide;
fonction grefferFilsDroit(ref s1:^sommet;ref s2:arbreBinaire):vide;
fonction grefferFilsGauche(ref s1:^sommet;ref s2:arbreBinaire):vide;
fonction elaguerFilsDroit(ref s1:^sommet):arbreBinaire;
fonction elaguerFilsGauche(ref s1:^sommet):arbreBinaire ;

Tas

fonction insérerValeur(ref T:tas de entier, val v:entier):vide;
fonction supprimerValeur(val T:tas de entier):entier;
fonction taille(val T:tas de entier):entier;
fonction creerTas(ref T:tas,val:v:entier):vide;

Table de hachage

fonction chercher(ref T:tableHash de clés, val v:clé):^clé;
fonction créerTableHachage(ref T: tableHash de clé,ref h:fonction):vide;
fonction ajouter(val x:clé, ref T:tableHash de clé):vide;
fonction supprimer(val x:clé, ref T:tableHash de clé):vide;

FIN