Licence **Sciences et Technologies**	**ANNEE : 2008/2009** **SESSION D'AUTOMNE 2008** **ETAPE :** CSB3, MHT53 **UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** 23 Décembre 2008 **Heure :** 8H30 **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 – Arbres binaires de recherche : 1.5 points
- Question 3 – B-Arbre : 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 et files : 2 Points
- Question 7 – Dictionnaire, tableaux et arbres binaires: 5 points

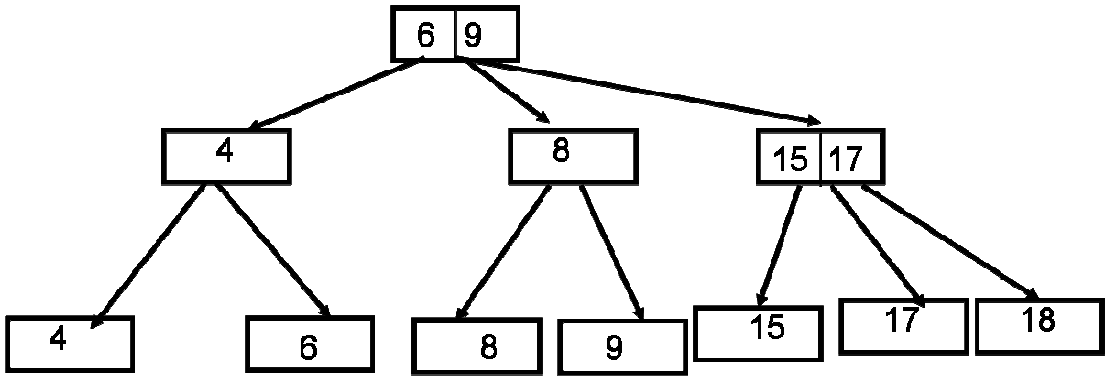
Question 1. Cochez les affirmations qui sont correctes :

☐ Le temps d'accès au dernier élément d'une liste doublement chainée est en O(n).
☐ Dans un arbre binaire de recherche le minimum est toujours la feuille la plus à droite dans l'arbre.
☐ Le temps d'accès à l'élément maximum d'un tas max est en O(1).
☐ Un arbre AVL est un arbre binaire de recherche.
☐ Une table de hachage à adressage ouvert utilise le type pointeur.
☐ Dans un tas, la primitive supprimerValeur consiste à supprimer la valeur stockée à la racine de l'arbre.
☐ La structure de B-arbre est une généralisation des AVL

☐ Si s est une structure dont un des champs est n : ^entier, on accède à l'entier par s^.n ?

Question 2. Soit la suite de clé 5,3,2,7,6,8,4,1. Construire l'arbre binaire de recherche 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 la suppression de la clé 3 sur l'arbre binaire de recherche ainsi construit.

Question 3. On donne le 2-3-arbre suivant :



Chaque question est à traiter sur l'arbre ci-dessus.

1 – Donnez sur l'arbre ci-dessus les positions des insertions qui conduiront à la création d'un trois nœud.

2 - Donnez l'arbre après insertion de la valeur 16.

3 - Dessinez et expliquez les modifications de l'arbre lors de la suppression de la valeur 9.

Question 4. Un concessionnaire de voitures d'occasion (spécialisé dans les voitures de moins de vingt ans) souhaite enregistrer son stock avec les informations suivantes pour chaque voiture : marque (20 caractères), modèle (20 caractères), année, un certain nombre d'options présentes sur la voiture, chaque option codée avec un nombre de 0 à 30. **Attention, dans ce qui suit on ne s'intéresse qu'à quelques questions concernant cette gestion et pas à la totalité de la gestion d'une concession.**

1 – Décrire la structure de donnée voiture.

2 – Quelle structure de données peut-on utiliser pour un ensemble de voiture afin de répondre en temps constant à la question :

« voitures de marque x Année y »

où les valeurs x et y sont fixés au moment de la requête (par exemple x=Renault, y =2000)

3 – Quelles structures de données peut-on utiliser pour accéder pour une option donnée à l'une des voitures les plus récentes ?

4 – Définir le type abstrait concession qui permet de décrire un ensemble de voitures et de répondre aux questions 2 et 3 ainsi que ses primitives.

5 – Ecrire la fonction *ajouterVoiture* qui est appelée chaque fois qu'une nouvelle voiture doit être répertoriée dans la concession. On ne traitera pas les cas d'erreurs.

6 – Ecrire les deux fonctions correspondantes aux questions 2 et 3.

Question 5.

1 Dessiner l'arbre AVL correspondant à la liste de clé (8,7,6,5,4,3). On donnera la liste des rotations avec le sommet de la rotation

2 Ecrire une fonction qui teste si un arbre binaire de recherche est un arbre AVL dans chacun des cas suivants :

- En utilisant des primitives d'arbre
- En utilisant les pointeurs.

Question 6. Ecrire une fonction qui prend en entrée une pile d'entier et qui modifie la pile en supprimant les entiers pairs et fournit en sortie la liste des entiers pairs supprimés. Par exemple, la pile [1,4,5,2,3,6,8] (où 1 est le fond de pile) est transformée en la pile [1,5,3] où (1 est le fond de pile) et la liste fournie est (4,2,6,8). Donnez la complexité de votre fonction.

Question 7. Lire l'ensemble du texte de la question AVANT de répondre. On pourra utiliser les fonctions vues en cours et en TD en précisant l'en-tête de la fonction et son fonctionnement. On considère une liste de mots au sens de la définition du cours. La structure définie est la suivante :

mot=structure
M :tableau[1..100] de caractère ; / les mots sont limités à la longueur 100 */*
Long :entier ;/ sa longueur*/*
finstructure
tout mot se termine par « \0 », la lettre « \0 » est plus petite que toute autre lettre.

1 - En précisant leur rôle, donnez une liste de primitives pour le type *mot*. On précisera l'en-tête de ces primitives.
2 – Implémenter la fonction concaténer qui colle deux mots l'un à la suite de l'autre et tronque le second mot si le tableau est saturé. Quelle est sa complexité ?
3 – Ecrire un opérateur *op::<* qui compare deux mots M1 et M2 et renvoit *vrai* si M1 est avant M2 dans l'ordre lexicographique (on écrira dans les algorithmes M1<M2).Quelle est sa complexité ?
4 – Ecrire une fonction *listeToDico* qui transforme une liste de mots en dictionnaire. Etudiez sa complexité dans le pire des cas?
5 – Ecrire une fonction *corrige* qui modifie la dernière lettre d'un mot du dictionnaire. On supposera que le mot se trouve dans le dictionnaire. Cette fonction prendra en argument un dictionnaire, le mot correct ainsi que la lettre à changer.
6 – On souhaite stocker des informations associées aux mots du dictionnaire de façon à y accéder rapidement. Pour un mot *m*, ces informations sont constituées d'une chaine de caractère donnant la définition et d'une liste de mots synonymes du mot *m*. Expliquez comment modifier la structure du dictionnaire.

Question bonus : Définir le type abstrait *informationPourDico* et donner la nouvelle définition de la structure de donnée dictionnaire. Donnez la liste des primitives.

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet);
fonction suivant(val L:liste d'objet);
fonction listeVide(val L:liste d'objet): booleen;
fonction créerListe(ref L:liste d'objet):vide;
fonction insérerAprès(ref L:liste d'objet; val x:objet;):vide;
fonction insérerEnTete(ref L:liste d'objet val x:objet):vide;
fonction supprimerAprès(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;

Listes doublement chainées (listeDC)

fonction finListe(val L:liste d'objet):vide;
fonction précédent(val L::liste d'objet): vide;

Piles

fonction valeur(ref P:pile de objet):objet;
fonction fileVide(ref P:pile de objet):booléen;
fonction créerPile(P:pile de objet) :vide
fonction empiler(ref P:pile de objet;val x:objet):vide;
fonction dépiler(ref P:pile de objet):vide;
fonction detruirePile(ref P:pile de objet):vide;

Files

fonction valeur(ref F:file de objet):objet;
fonction fileVide(ref F:file de objet):booléen;
fonction créerFile(F:file de objet);vide;
fonction enfiler(ref F:file de objet;val x:objet):vide;
fonction défiler (ref F:file de objet):vide;
fonction detruireFile(ref F:file de objet):vide;

Arbres binaires

fonction getValeur(val S:sommet):objet;
fonction filsGauche(val S:sommet):sommet;
fonction filsDroit(val S:sommet):sommet;
fonction pere(val S:sommet):sommet;
fonction setValeur(ref S:sommet;val x:objet):vide;
fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
fonction supprimerFilsGauche(ref S:sommet):vide;
fonction supprimerFilsDroit(ref S:sommet):vide;
fonction detruireSommet(ref S:sommet):vide;
fonction créerArbreBinaire(val Racine:objet):sommet;

Arbres planaires

fonction valeur(val S:sommetArbrePlanaire):objet;
fonction premierFils(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction frere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction pere(val S:sommetArbrePlanaire):sommetArbrePlanaire;
fonction créerArborescence(val racine:objet):sommetArbrePlanaire;
fonction ajouterFils(ref S:sommetArbrePlanaire,val x:objet):vide;
fonction supprimerSommet(ref S:sommetArbrePlanaire):vide;
fonction créerArbreBPlaniare(val Racine:objet):sommet;

Arbres binaire de recherche

fonction ajouter(ref A:arbreBinaire d'objets, val v:objet):vide;
fonction supprimer(val A: arbreBinaire d'objets, val v:objet):vide;

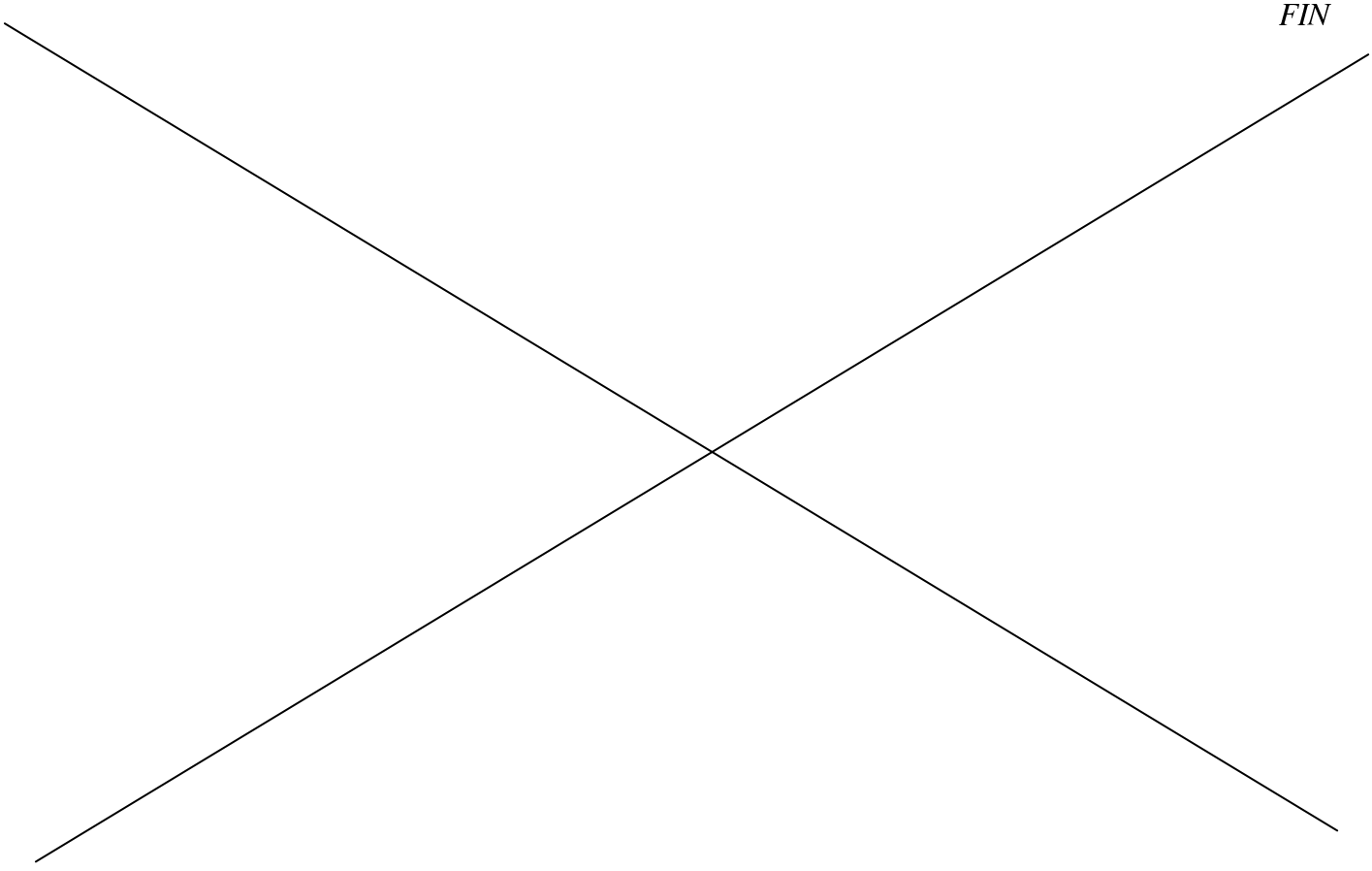
Tas

fonction valeur(ref T:tas d'objet): objet;
fonction ajouter(ref T:tas de objet, val v:objet):vide;
fonction supprimer(val T:tas de objet):vide;
fonction creerTas(ref T:tas,val:v:objet):vide;
fonction detruireTas(ref T:tas):vide;

Dictionnaire

fonction appartient((ref d:dictionnaire,val M::mot):booléen;
fonction creerDictionnaire(ref d: dictionnaire):vide ;
fonction ajouter(ref d:dictionnaire,val M::mot):vide;
fonction supprimer(ref d:dictionnaire,val M:mot):vide;
fonction detruireDictionnaire(ref d:dictionnaire):vide;

Table de Hachage

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

FIN