DISVE Pôle Licence	**ANNEE UNIVERSITAIRE 2010/2011** **SESSION 1 D'AUTOMNE** **PARCOURS / ETAPE :** CSB3, MHT53 **Code UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** 20 Décembre **Heure :** 8H30 **Durée :** 1H30 Documents : non autorisés Epreuve de M/Mme : DELEST **Vous devez répondre directement sur le sujet qui comporte 8 pages. Insérez ensuite votre réponse dans une copie d'examen comportant tous les renseignements administratifs**	

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

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

Barème indicatif

- Question 1 – Connaissances générales : 2 points
- Question 2 – Containeur liés aux arbres binaires de recherche : 6 points
- Question 3 – Connaissance des structures de données : 6 Points
- Question 4 – Utilisation des structures de données : 6 points

Question 1. Cochez les affirmations correctes et quelle que soit la réponse justifiez sur la ligne en pointillé.

☐ Dans un tas MIN le minimum est toujours à la racine.

……

☐ Une file de priorité est un arbre binaire de recherche.

……

☐ La structure de pile permet l'accès au dernier élément entré dans le containeur.

……

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

……

Question 2. Soit la liste de clé A=(6,2,4,1,8,3,7,5).

1 – Ecrire la fonction *ajouter* qui insère un élément dans un tas MAX.

2 - Construire le tas MAX T correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

3 - Montrez l'exécution de la suppression sur l'arbre T.

3 – Construire l'arbre binaire de recherche B correspondant à l'insertion consécutive des clés de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

4 – Donner la liste des sommets en ordre préfixe, en ordre infixe et en ordre suffixe.

5 – Construire l'AVL correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final.

Question 3. Soit une suite de clé dans un tableau T. On considère un tableau de dimension N contenant des entiers appartenant à l'ensemble {0,1}. Par exemple, le tableau de dimension 8 est donné par T[1]=0, T[2]=1, T[3]=0, T[4]=0, T[5]=1, T[6]=0, T[7]=1, T[8]=1.

1 – Ecrire la fonction *verifier* qui vérifie que les éléments du tableau appartiennent à l'ensemble {0,1} et que le nombre de zéro est le même que le nombre de 1.

2 – En utilisant les primitives du type abstrait listeSC, écrire la fonction *tableauListe* qui a pour paramètre un tableau d'entier T et fournit en sortie une liste simplement chainée L dans le même ordre que le tableau d'entrée (sur l'exemple, L= (0,1,0,0,1,0,1,1).

3 - Ecrire cette même fonction *tableauListe* en utilisant l'implémentation dynamique sans appel aux primitives du type abstrait listeSC.

4 – Ecrire une fonction *facteur01* qui à partir de la liste L simplement chainée fournit la pile des indices j tels que T[j]=0 et T[j+1]=1 dans l'ordre inverse (sur l'exemple [6,4,1[où 1 est le sommet de pile).

5 – Ecrire une fonction *extraitPair* qui extrait de la pile toutes les positions ayant un numéro pair et les range dans un file dans un ordre croissant sans changer la pile (sur l'exemple [4,6[où 4 est le premier de file).

6 – On souhaite écrire une fonction *supZeroPair* qui supprime dans la liste L, les valeurs 0 qui correspondent à des indices pairs dans el tableau (sur l'exemple, la nouvelle liste est (0,1,0,1,1,1)). La structure de liste simplement chainée n'est pas adaptée. Pourquoi ? Quelles sont les modifications à apporter dans à la fonction *tableauListe* ? Ecrire la fonction *supZeroPair*.

Question 4. Pour éviter la paralysie d'une ville en cas d'intempérie, une ville décide de mettre en place un système permettant que chaque véhicule de transport en commun puisse transmettre une information comportant géolocalisation (une latitude et une longitude) ainsi que un signal correspondant à l'état de la circulation : bloqué, ralentie, normale. La latitude et la longitude sont données par un triplet (degré, minute, seconde). Chaque véhicule est immatriculé par une suite de 8 caractères. On s'intéresse uniquement à ce que cette information soit accessible en temps réel pour les usagers sans prendre en compte cartographie ou routage optimal des usagers. Il est plus important pour un usager de savoir que le trafic est bloqué plutôt que normal.

Dans la suite, on décrira précisément chaque structure de donnée. On précisera s'il s'agit d'un containeur. On ne demande pas d'écrire les primitives mais uniquement de donner l'en-tête de la primitive et sa fonction.

1 – Décrivez la structure de données permettant de stocker pour un véhicule de transport donné les informations transmises. En donner les primitives.

2 – Décrivez la structure de données permettant de mettre à jour les informations liées à un véhicule donné. En donner les primitives.

3 - Quelle structure de données peut-on utiliser afin qu'un utilisateur qui connaît la géolocalisation de l'endroit où il se rend puisse consulter l'état du trafic ? On ne demande pas les primitives.

Listes simplement chainées (listeSC)

fonction valeur(val L:liste d'objet):objet;
fonction debutListe(val L:liste d'objet) :vide ;
fonction suivant(val L:liste d'objet) :vide ;
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;

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;

File de priorité

fonction changeValeur(ref T:tas d'objet,val s:sommet,val v:objet):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