UNIVERSITE BORDEAUX 1 Sciences Technologies DISVE Pôle Licence	**ANNEE UNIVERSITAIRE 2009/2010** **SESSION 1 D'AUTOMNE** **PARCOURS / ETAPE :** CSB3, MHT53 **Code UE :** INF 251 **Epreuve :** Algorithmes et Structures de Données Fondamentaux **Date :** xxx Juin **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**	Département D L Licence

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 arbre binaire de recherche le maximum est toujours placé sur une feuille.

Non par exemple la séquence (4,3) conduit à un maximum à la racine.

☐ Un arbre AVL de taille 15 n'est jamais un arbre binaire de recherche.

Par définition un AVL est un arbre binaire de recherche équilibré (c'est-à-dire dont la hauteur est aussi proche que possible de $\log_2(n)$ où n est la taille de l'arbre) donc c'est un arbre binaire de recherche.

☒ Dans un B-arbre les valeurs se trouvent uniquement sur les feuilles.

Les valeurs sur les nœuds internes servent uniquement à « aiguiller » l'insertion.

☒ Si s est une structure dont un des champs x est un pointeur d'entier, on accède à l'entier par s.x^.

s.x désigne le pointeur donc en mettant ^, on accède à l'entier

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

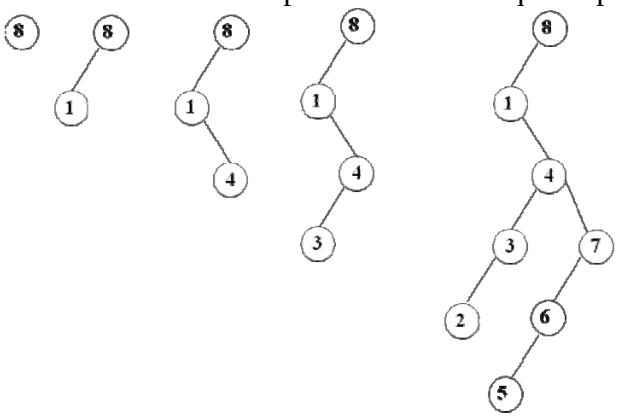
1 – Ecrire la fonction *supprimer* qui supprime un élément dans un arbre binaire de recherche.

On suppose que le sommet contient l'élément à supprimer.

```
fonction supprimer(ref x:sommet):booléen;
    var p,f,y:sommet;
    début
        si estFeuille(x) alors
            p=pere(x);
            si filsGauche(p)==x alors
                supprimerFilsGauche(p)
            sinon
                supprimerFilsDroit(p)
            finsi
        sinon
            f=filsDroit(x)
            si f!=NIL
                y=cherchePlusPetit(f);
            sinon
                f=filsGauche(x);y=cherchePlusGrand(f);
             finsi
            v=getValeur(y);supprimer(y);setValeur(x,v);
         finsi
```

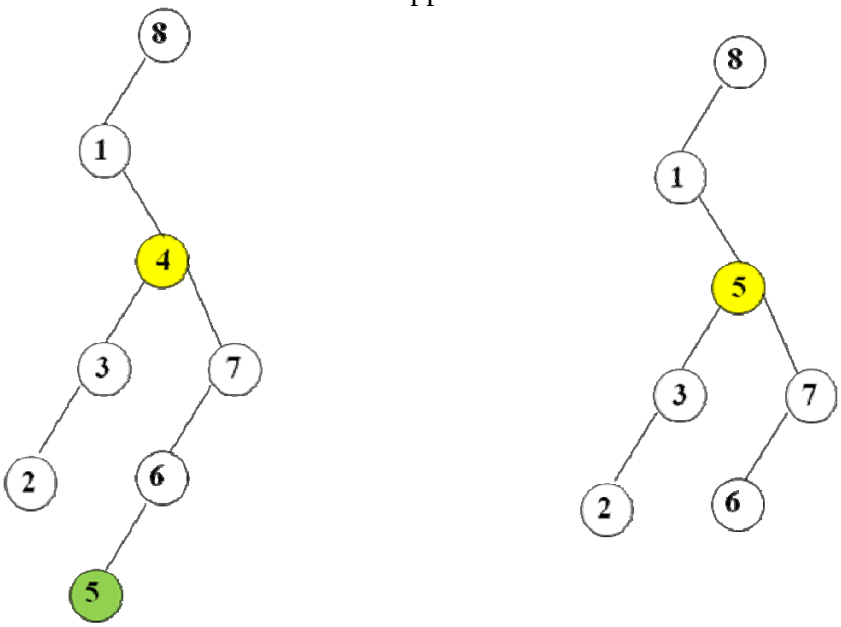
fin

2 - 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.

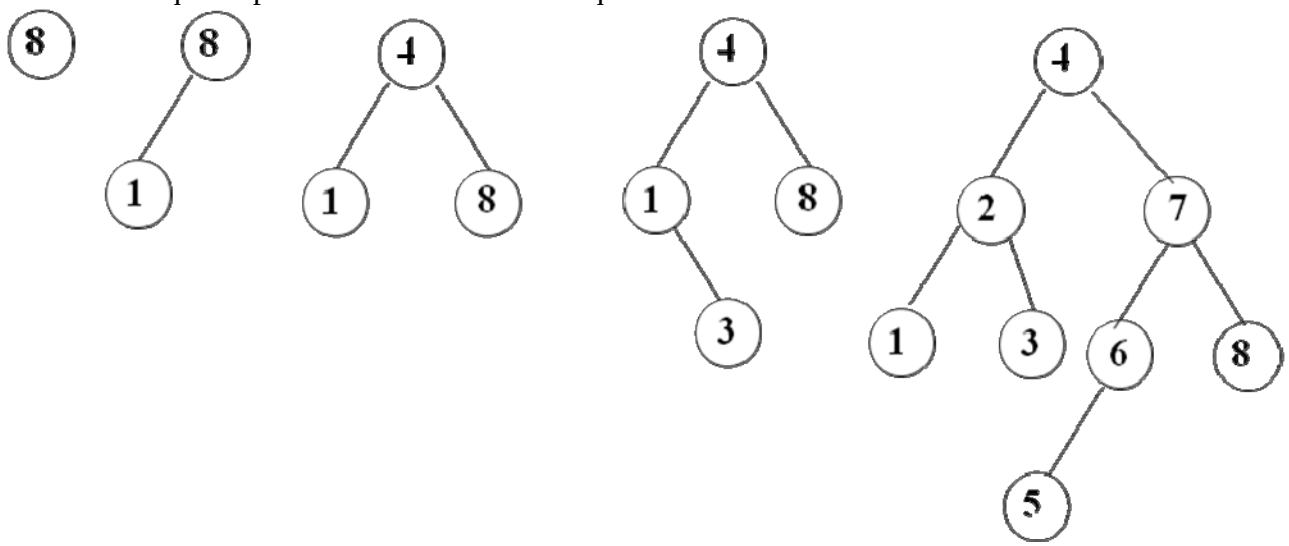


3 – Donnez pour l'arbre B la liste des sommets en ordre préfixe, en ordre infixe et en ordre suffixe.
Ordre préfixe : 8,1,4,3,2,7,6,5
Ordre infixe : 1,2,3,4,5,6,7,8
Ordre suffixe : 2,3,5,6,7,4,1,8

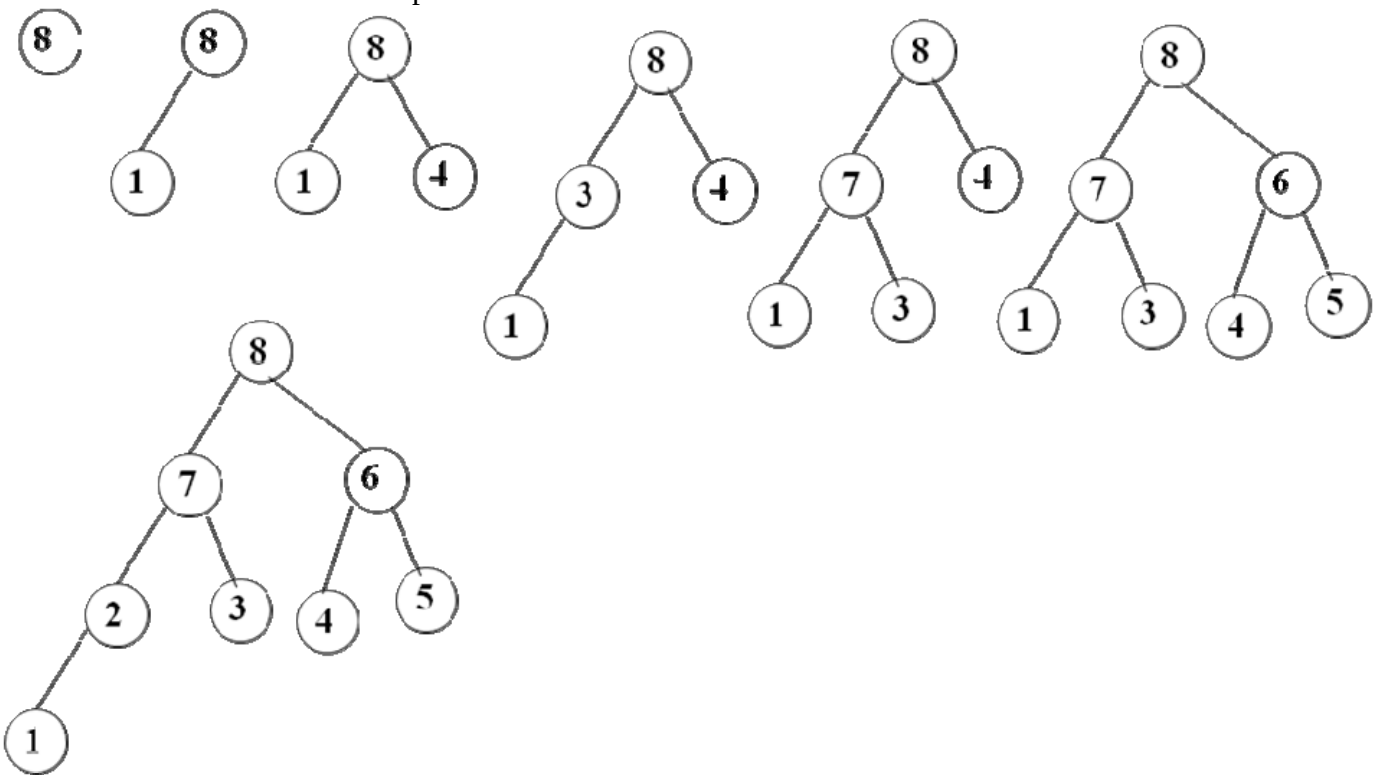
4 - Montrez l'exécution de la suppression de la clé 4 sur l'arbre B.



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.



6 – Construire le tas Max correspondant à l'insertion consécutive de la liste A. On dessinera l'arbre final.



Question 3. Soit une suite de clé de l'ensemble {0,1} stockées dans une liste L. Par exemple L=(0, 0,1,0,0,1,1,1).

1 – Ecrire la fonction booléenne *verifier* qui retourne vrai quand dans la liste L, il y a autant de valeurs 0 que de valeurs 1. Elle retourne faux dans le cas contraire. On utilisera les primitives du type abstrait listeSC.

```
fonction verifier(ref L : listeSC d'entier) :booléen;
    var cpt :entier
    début
        cpt=0 ;
        debutListe(L) ;
        tantque !finliste(L) faire
            si valeur(L)==0 alors
                cpt=cpt+1 ;
            sinon
                cpt=cpt-1 ;
            finsi
        fintantque
        retourner(cpt==0)
    fin
```

2 – On souhaite disposer d'une fonction *complete* qui complète la liste L lorsque la fonction *vérifier* retourne faux. Est-il plus facile de réaliser cette fonction si la liste est doublement chainée que si elle est simplement chainée ?

Non puisque les valeurs 0 ou 1 peuvent être ajoutées en tête après comptage.

3 – Ecrire la fonction *complete* en utilisant le type abstrait listeDC.

```
fonction complet(ref L : listeDC d'entier) :vide;
    var cpt :entier
    début
        cpt=0 ;
        debutListe(L) ;
        tantque !finliste(L) faire
```

```
            si valeur(L)==0 alors
                cpt=cpt+1 ;
            sinon
                cpt=cpt-1 ;
            finsi
        fintantque
        si cpt<0 alors
            completeTete(L,1,-cpt)
        sinon
            completeTete(L,0,cpt)
        finis
    fin
fonction completeTete(ref L : listeSC d'entier, val x,c :entier) :vide;
    var i :entier ;
    début
        pour i allant de 1 à c faire
                ajouterEnTete(L,x)
        finpour
    fin
```

4 – Ecrire cette même fonction *complete* en utilisant l'implémentation dynamique à la place des primitives.

```
cellule=structure
            valeurElement:entier;
            pointeurSuivant:^cellule;
         finstructure;
listeSC d'entier=structure
          premier:curseur;
          cle:curseur;
        finstructure
fonction complet(ref L : listeSC d'entier) :vide;
    var cpt :entier ;
    début
        cpt=0 ;
        L.cle=L.premier;
        tantque L.cle !=NIL faire
            si L.cle^.valeurElement==0 alors
                cpt=cpt+1 ;
            sinon
                cpt=cpt-1 ;
            finsi
            L.cle=L.cle^. pointeurSuivant ;
        fintantque
        si cpt<0 alors
            completeTete(L,1,-cpt)
        sinon
            completeTete(L,0,cpt)
        finis
    fin
fonction completeTete(ref L : listeSC d'entier, val x,c :entier) :vide;
    var i :entier ;
    var tmp :curseur ;
    début
```

```
        pour i allant de 1 à c faire
            new(tmp) ;
            tmp^. valeurElement =x ;
            tmp^. pointeurSuivant=L.premier ;
            L.premier=tmp ;
        finpour
    fin
```

5 – Ecrire la fonction *separe* qui à partir d'une liste doublement chainée fournit 2 structures :

- une pile contenant autant de 0 que ceux contenus dans la liste,
- une file contenant autant de 1 que ceux contenus dans la liste.

```
fonction separe(ref L : listeDC d'entier ; ref P :pile d'entier ; ref F : file d'entier) :vide;
    début
        debutListe(L) ;
        creerPile(P) ;
        creerFile(F) ;
        tantque !finliste(L) faire
            si valeur(L)==0 alors
                empiler(P,0) ;
            sinon
                enfiler(F,1) ;
            finsi
        fintantque
    fin
```

Question 4. Un magasin a une seule porte d'entrée, il y a plusieurs vendeurs et une seule caisse pour payer. Les clients rentrent tous par la porte d'entrée, ils attendent pour être pris en charge par un vendeur, puis, après avoir été servis, ils vont à la caisse où ils attendent leur tour pour payer. Parfois un client excédé par l'attente quitte le magasin.

1 – Quelle structure de donnée permet de modéliser la plupart des éléments ci-dessus ? Pourquoi ?

Une file car en général le premier arrivé est le premier servi. La seule opération qui n'est pas modélisable directement par une file est le client qui quitte la file d'attente. Pour « retirer » le client il faudra défiler la file dans une autre file.

2 – Ecrivez les structures de données client, vendeur, caisse, magasin.

client : entier (comme dans les organismes ou on prend un ticket à l'entrée, il faudra donc un compteur initialisé à 1 lors de l'ouverture du magasin)

vendeur : tableau[1..N] d' entier (il y a N vendeurs, c'est le numéro du client qui est en cours de traitement qui est affecté)

caisse =file de client ;

magasin =file de client ;

3 - Décrivez le(s) type(s) abstrait(s) correspondant à ces structures et tout particulièrement les primitives.

unMagasin=structure

compteur :entier ; / pour affecter un numéro aux clients*/*

lesVendeurs :vendeur ;

laCaisse :caisse ;

entree :magasin ;

finstructure

Cette structure est un containeur il y a donc 4 primitives :

- creerUnMagasin

- détruireUnMagasin

-ajouterClientMagasin / le client se présente à la porte d'entrée */*

-enleverClientMagasin/ le client quitte le magasin en passant par la caisse */*

On peut également mentionner les primitives suivantes

-affecterClientAVendeur / le premier client de la file entree est affecté à un vendeur */*

-affecterClientACaisse / le vendeur a fini de servir un client il se présente à la caisse */*

-departPrecipiteClient / il faut défiler la file entree pour le retirer de la file */*

On notera que si un client part sans acheter, il suffit d'affecter un nouveau client au vendeur.Il faudra traiter le cas ou la file entree est vide.

4 - Ecrire les primitives d'accès et de modification du type vendeur ?

Comme il s'agit d'un tableau, il va falloir l'initialiser. On peut par exemple initialiser tous les éléments à -1.

```
fonction initVendeur(ref V :tableau[1..N] d'entiers) ;
    var i :entier ;
    début
        pour i allant de 1 à N faire
            V[i]=-1
        finpour
    fin
```

Les fonctions ci-dessous ne sont pas des primitives mais modifient les valeurs des variable de type vendeur au sein d'un magasin.

```
fonction affecterClientAVendeur (ref U :unMagasin:) :booleen ;
    var i ,C:entier ;
    début
        i=1 ;
        tantque  V[i ]!=-1 et i<n faire
            i=i+1 ;
        fintantque ;
        si V[i]==-1 alors
            V[i]= valeur(U.entree) ;
            defiler(U.entree)
```

```
            retourner(vrai)
        sinon
            retourner(faux)
        finsi
    fin
fonction affecterClientACaisse (ref U :unMagasin , val v :entier) :vide ;
    début
    enfiler(U.laCaisse,U.lesVendeurs[v]) ;
        U.lesVendeurs[v])=-1 ;
    fin
```

5 – Si ce magasin est muni d'un système de carte de fidélité, une fonctionnalité de la caisse consiste à proposer la saisie du nom du client. Quelle structure de donnée peut permettre de gérer cette fonction ? Pourquoi ? Modifier le type abstrait de la question 3 en conséquence. On ne demande pas d'écrire les primitives.

On peut choisir un arbre binaire de recherche dont les nœuds sont les numéros de carte fidélité. Parfois le client oublie sa carte de fidélité, un dictionnaire permet d'accéder rapidement aux informations concernant le client par le nom. On a donc une nouvelle structure de donnée qui regroupe les informations de la journée (unMagasin) et ces deux structures.

toutUnMagasin=structure
M :unMagasin ;
F : arbreBinaire d'entier ;
D :dictionnaire de lettres ;
finstructure

chaque cellule de l'arbre ainsi que la feuille du dictionnaire pointeront vers l'ensemble des données du client (adresse, téléphone, nomre de points ….).Il y aura donc des primitives d'initialisation : création, destruction, recherche d'un client par numéro.

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