| | |
|---|---|
| **Licence**<br>**Sciences et Technologies** | **ANNEE : 2007/2008** **SESSION DE MARS 2008**<br><br>**ETAPE :** **UE :** INF 251<br>**Epreuve :** Algorithmes et Structures de Données Fondamentaux<br>**Date :** 7 Mars 2008 **Heure :** 14H00 **Durée :** 3 H<br>**Documents :** Tous documents interdits<br>**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**<br>**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 – ABR : 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 files et piles : 2 Points
- Question 7 – Listes et tas: 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 tas, le minimum est toujours à la racine de l'arbre.

☒ Le temps moyen d'accès à l'élément maximum d'un arbre binaire de recherche est en $O(\log_2(n))$.

☒ Un arbre tas est un arbre binaire.

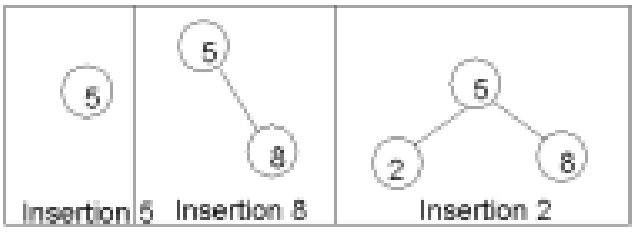
☒ Pour un tas donné l'obtention de la liste des nombres triés est en temps O(n).

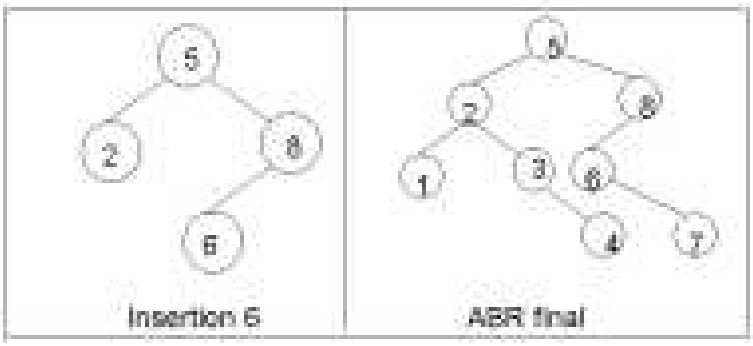
☒ Dans un arbre binaire de recherche, la primitive insertion consiste à ajouter une feuille valuée à l'arbre.

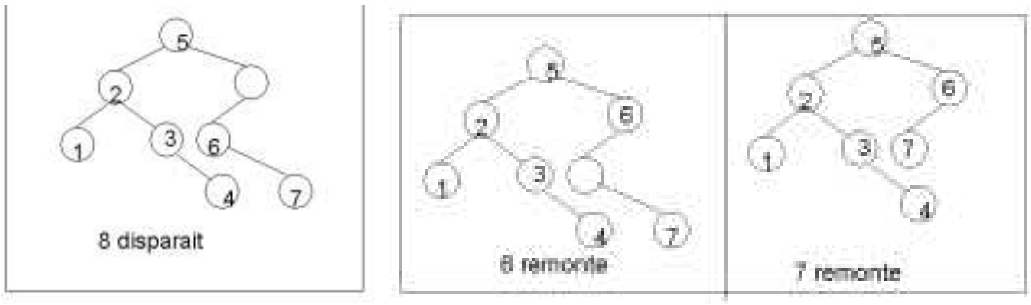
€ Dans une table de hachage chainée, la collision est résolue en utilisant une fonction de hachage.

☒ Un arbre 2-3 est un arbre équilibré éventuellement vide tel que tout noeud à 2 ou 3 fils

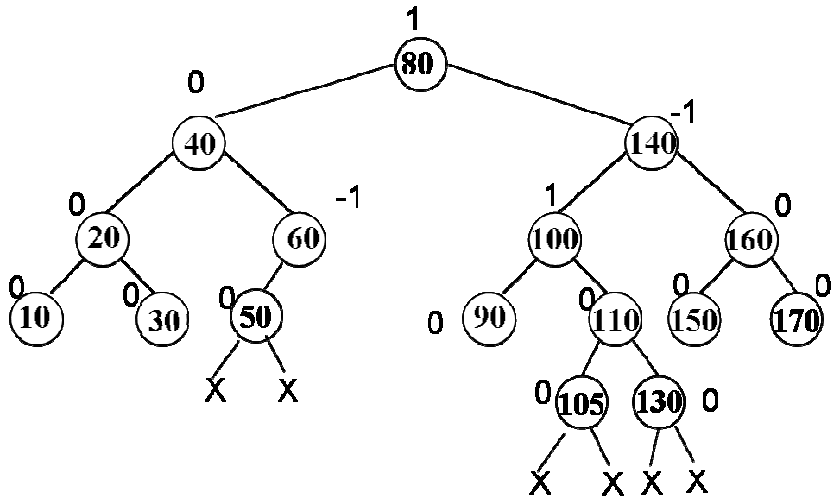
**Question 3.** Soit la suite de clés 5,8,2,6,3,4,1,7. Construire le l'arbre binaire de recherche correspondant à l'insertion consécutive des clés, on dessinera l'arbre après chacune des quatre premières insertions ainsi que l'arbre final. Montrez l'exécution de la suppression de la valeur 8 sur l'arbre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






**Question 3.** On donne l'arbre AVL suivant ;

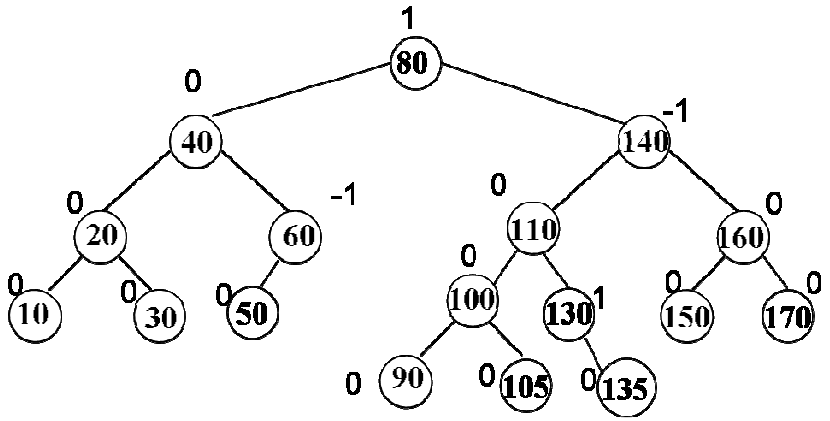



1 – Donnez sur l'arbre ci-dessus les positions des insertions de feuille qui conduiront à un arbre déséquilibré ?

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

3 - Dessinez la suite des modifications de l'arbre lors de l'insertion de la valeur 135.

*La valeur 135 s'insère comme fils droit du nœud contenant 130. Par suite, l'arbre est déséquilibré. Le facteur d'équilibrage du nœud contenant 110 passe à 1, celui correspondant à 100 passe à 2. Le nœud pour lequel le facteur de déséquilibrage est mauvais est le nœud contenant la valeur 100 (facteur d'équilibrage 2). On effectue une rotation simple à gauche autour du nœud 100. Après cette opération l'arbre est AVL.*




**Question 4.** On souhaite enregistrer pour une gare donnée l'ensemble des trains avec les informations suivantes : provenance ou destination (30 caractères), heure de départ ou d'arrivée (heure,minute), retard estimé (minutes), voie (entier).

1 – Décrire la structure de donnée train.

*Train=structure*

*Ville : tableau[1..30]de car ;*
*Provenance :booléen ; /* vrai si le train vient de la ville faux si c'est sa destination*/*
*Heure :entier ;*
*Minute :entier ;*
*Retard :entier ;*
*Finstructure ;*

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

« Trains en provenance de x à y Heure »

où les valeurs x et y sont fixés au moment de la requête (par exemple x=Arcachon, y =10)

*On choisit une table de hashage dont la clé est constituée par les champs provenance et heure. Si on choisit un adressage ouvert on a donc une table de hachage de Train.*

3 – Quelle structure de donnée supplémentaire peut-on utiliser pour accéder en temps constant au train qui part le plus tôt quelle que soit sa destination.

*On choisit un tas Min la valeur d'insertion sera constituée par la valeur heure*60+minute. On stockera dans le nœud l'adresse du train dans la table de hashage.*

4 – Donner la structure ensembleTrain qui permet de décrire un ensemble de trains répondant aux deux questions précédentes.

*ensembleTrain=structure*

*H :tableHashage de train ;*
*A :tas de clés ;*
*Finstructure ;*

5 – Ecrire la fonction *ajouterTrain* qui est appelée chaque fois qu'un train doit être répertorié dans l'ensemble des trains. On ne traitera pas les cas d'erreurs.

*Dans les primitives de tas, il faut changer*

*valeurSommet(T.tas,x) par valeurSommet(T.tas,x)^.heure*60+ valeurSommet(T.tas,x)^.minute.*

```
Fonction ajouterTrain(val T :train, ref E :ensembleTrain) :vide ;
   Var p :^train ;
   Début
      ajouter(T,E.H)) ;
      p =Chercher(E.H,T) ;
      insererValeur(E.A,p) ;
   Fin
```

6 – Ecrire la fonction *supprimerTrain* qui est appelée chaque fois qu'un train est arrivé ou est parti de la gare. On ne traitera pas les cas d'erreurs.

*Dans les primitives de tas, il faut changer*

*valeurSommet(T.tas,x) par valeurSommet(T.tas,x)^.heure*60+ valeurSommet(T.tas,x)^.minute.*

```
Fonction supprimerTrain (val T :train, ref E :ensembleTrain) :vide ;
   Var p :^train ;
   Début
      p =Chercher(E.H,T) ;
      supprimer(H.T,p) ;
      supprimerValeur(E.A,p) ;
   Fin
```

**Question 5.** On définit la longueur de cheminement interne d'un arbre binaire comme la somme des hauteurs des sommets interne de l'arbre (nombre d'arêtes de la racine au sommet). Ecrire la fonction cheminementInterne qui prend en paramètre un arbre et calcule cette longueur en utilisant uniquement les primitives du type abstrait.

```
Fonction cheminementInterne(ref A :arbreBInaire ;val h :entier) :entier
   Var c :entier ;
   Début
      Si estFeuille(A) alors
          retourner(0)
      sinon
         c=h ;
         si filsGauche(A) !=NIL alors
            c=c+ cheminementInterne (filsGauche(A),h+1)
         finsi
         si filsDroit(A) !=NIL alors
            c=c+ cheminementInterne (filsDroit(A),h+1)
         finsi
         retourner( c)
      finsi
   fin
```

*L'appel se fait avec cheminementInterne (A,0).*

**Question 6.** Ecrire une fonction qui prend en entrée une file d'entier et qui fournit en sortie une file telle que tous les nombres multiples de 3 se retrouvent en tête de la file et dans le même ordre que la file originale, les autres éléments sont dans la file en ordre inverse. Par exemple, [1,3,5,4,2,6,8[ fournit en sortie en [3,6,8,2,4,5,1 [.

```
Fonction mul3File(val  F :file de entier) :file de entier ;
   var FC :file d'entier ;
   var P :  pile d'entier ;
   var v : entie ;
   Début
      creerPile(P) ;
      creerFile(FC) ;
      tantque !fileVide(F)) faire
         v=valeur(F) ;
         si v  est multiple de 3 alors
            enfiler(FC,v) ;
         sinon
            empiler(P,v) ;
         finsi
         défiler(F) ;
```

```
    fintantque
    tantque !pileVide(P) faire
        enfiler(FC,valeur(P)) ;
        depiler(P)
    fintantque
    retourner(FC)
fin
```

**Question 7.** On considère un tableau de dimension N contenant des nombres pairs. Par exemple, le tableau de dimension 8 contient 4,8,16,6,10,24,10,26.

1 – Ecrire une fonction *tableauToListe* qui transforme un tel tableau en une liste simplement chainée des nombres divisés par 2. Cette fonction renvoie vrai si les éléments du tableau sont conformes à l'énoncé (les entiers sont pairs) et faux sinon.

```
Fonction tableauToListe(val  T :tableau[1..N]  de entier,ref L : listeSC de entier ):booléen;
    var p :^entier ;
    var i :entier ;
    Début
        Pour i allant de 1 à N faire
            si !estPair(T[i]) alors
                retourner(faux)
            finsi
        finpour
        creerListe(L) ;
        insererEnTete(T[1]/2,L) ;
        p =premier(L)   ;
        Pour i allant de 1 à N faire
            insererApres(T[i]/2,L,p) ;
            p=suivant(L,p)
        finpour
        retourner(vrai)
    fin
```

2 – Soit M la moyenne des nombres du tableau. Ecrire une fonction *supMoyenne* qui à partir de la liste ainsi constituée calcule la moyenne et fournit la liste des valeurs T[j] telles que T[j]<M<T[j+1] pour j<N dans l'ordre inverse (sur l'exemple [5,5,4]).

```
fonction supMoyenne (ref L :listeSC d'entier,val M :réel) :listeSC d'entier;
    Var SP,P :^cellule ;
    Var pic :listeSC d'entier ;
   Début
        creerListe(pic) ;
        Si ! listeVide(L)
            P=premier(L) ;
            SP=suivant(L,P)
            Tantque SP !=NIL faire
                Si contenu(P)<M  et contenu(SP)>M alors
                    insererEnTete(contenu(SP),pic)
                finsi
                P=SP ;
                SP= suivant(L,SP);
            Fintantque
        Finsi
        Retourner(pic)
    Fin
```

3 – Si on choisit d'utiliser des listes doublement chainées à la question 1, écrire la fonction *supMoyenneDC*.

*La fonction est inchangée car on ne fait pas référence au précédent on peut cependant utiliser la fonction dernier.*

```
fonction supMoyenneDC(ref L :listeDC d'entier,val M :réel) :listeSC d'entier;
    Var SSP,P :^cellule ;
    Var pic :listeSC d'entier ;
   Début
       creerListe(pic) ;
       Si ! listeVide(L)
          P=premier(L) ;
          Tantque Dernier(L) !=P  faire
             SP=suivant(L,P)
             Si contenu(P)<M  et contenu(SP)>M alors
                insererEnTete(contenu(SP),pic)
             finsi
             P=SP ;
          Fintantque
       Finsi
       Retourner(pic)
    Fin
```

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

- En temps

*Les deux fonctions sont équivalentes en temps : une boucle tantque qui parcourt une liste de taille N.*

- En mémoire

*Les deux fonctions ne sont pas  équivalentes la fonction en listeSC utilise moins de mémoire par définition que celle en listeDC puisque une cellule stocke un pointeur de plus. Cependant en ce qui concerne la complexité,  elles sont équivalentes, O(n).*

- En lisibilité de la fonction.

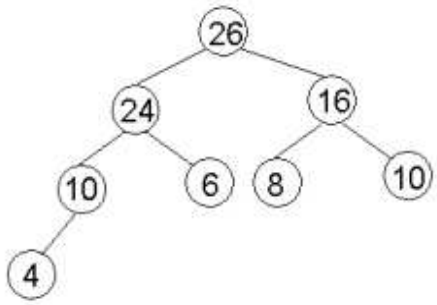
*Il y a un avantage très léger pour la fonction listeDC. En effet, au niveau de l'arrêt de boucle, on comprend mieux les comparaisons qui sont effectuées grâce à la primitive dernier.*

5 – Ecrire une fonction *listeToTas* qui transforme la liste de la question 1 en un tasMax.

```
fonction insérerValeur(ref T:tas de entier, val v:entier):vide
  début
    T.dernier=T.dernier+1;
    T.tas.arbre[T.dernier]=v;
    reorganiseTasMontant(T,T.dernier);
  fin
finfonction
fonction reorganiseTasMontant(ref T: tas,val x:sommet):vide;
  var p:sommet;
  var signal:booléen;
  début
    p=père(T.tas,x);
    signal=vrai;
    tantque x!=racine(T.tas) et signal faire
      si valeurSommet(T.tas,x)<valeurSommet(T.tas,p) alors
        échanger(T.tas.arbre[p],T.tas.arbre[x])
        x=p;
        p=père(T.tas,x);
      sinon
        signal=faux
      finsi
    fintantque
  fin
fin
```

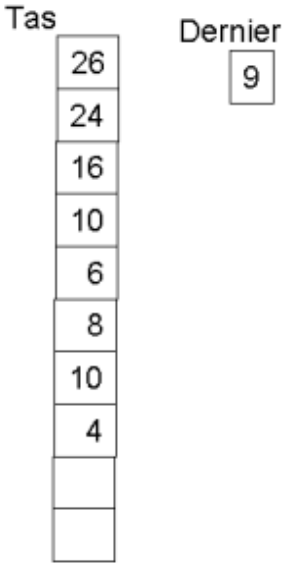
```
fonction listeTas(ref L:liste d'entier):tas;
  var T:tas;
  car x:^cellule;
  début
    creerTas(T,contenu(premier(L)));
    x=suivant(L,premier(L));
    tantque x!=NIL faire
      insérerValeur(T,contenu(L,x));
      x=suivant(L,x);
    fintantque
    retourner(T)
  fin
```

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).




PR=26,24,10,4,6,16,8,10
IN=4,10,24,6,26,8,16,10
SU=4,10,6,24,8,10,16,26

7 - Donner sous forme de schémas ou de tableaux, la représentation des structures de données dans le cas où l'arbre est implémenté en allocation statique




8 – Ecrire en utilisant les primitives du type abstrait *arbreBinaire* la fonction *compteNoeudTA* dont le résultat est le nombre de nœud dans un tas ayant une valeur supérieure à la moyenne M.

```
fonction compteNoeudTA(ref A:arbreBinaire de entier ; val M :réel):entier;
  val c :entier,
  début
    Si valeurSommet(A) >M alors
        c= 1
    sinon
        c= 0
    finsi
    Si filsGauche(A) !=NIL  alors
      c=c+compte1NoeudTA(filsGauche(A)) ;
```

```
    finsi
    Si filsDroit(A) !=NIL alors
        c=c+compte1NoeudTA(filsDroit(A)) ;
    Finsi
    Retourner(c) ;
Fin
```

9– Ecrire la fonction *compteNoeudST* en utilisant directement l'implémentation en allocation statique.

```
fonction compteNoeudTA(ref A:arbreBinaire de entier ; val M :réel):entier;
    val c,i :entier,
    début
        c= 0
        pour i allant de 1 à T.dernier  faire
            si A .arbre[i]>M alors
                c=c+1 :
            finsi
        finpour
        retourner(c) ;
    fin
```

10 – Donnez les avantages et inconvénients des implémentations des fonctions *compteNoeudTA* et *compte1NoeudST.*

- En temps

*Les deux fonctions sont équivalentes en complexité. Cependant la fonction compte1NoeudST est meilleure car il n'y a pas la gestion des appels récursifs.*

- En mémoire

*Les deux fonctions ne sont pas équivalentes la fonction en compte1NoeudST utilise moins de mémoire par définition que celle en compteNoeudTA puisque la gestion du parcours du tas en terme de mémoire sera de l'ordre de* $O(log_2(n))$.

**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*