Algorithmique 1

DS 4

Documents **non** autorisés

NB : Dans toutes les fonctions demandées, il faudra utiliser uniquement les primitives (aucune référence à la structure). Ces primitives sont rappelées en annexe

Exercice 4.1

1. Soit H la hauteur d'un tas. Justifier pourquoi la taille d'un tas (nombre d'éléments stockés) est toujours inférieur ou égal à $2^{H+1} - 1$.
2. Ecrire une fonction qui vérifie si un arbre binaire est bien un tas-min.
3. Quelle est la complexité de votre fonction ?

Exercice 4.2

On considère le type `dico`. Ecrire une fonction qui détermine le mot de longueur maximale d'un dictionnaire passé en paramètre.
Pour vous faciliter la mise en oeuvre, Vous allez d'abord

1. écrire une fonction `position_mot_max( ref D : dico)` qui retourne un curseur (pointeur) sur la fin du mot de longueur maximale.
2. utiliser le résultat de la fonction précédente pour stocker la suite de caractères dans une liste simplement chainée qui sera retournée comme résultat de la fonction `mot-longueur-max( ref D : dico )` .

Exercice 4.3

1. Partant d'un arbre binaire vide, construire un AVL en utilisant la fonction d'insertion(vue en cours) appliquée à la suite des clés suivantes :

   $$342, 206, 444, 523, 607, 301, 142, 183, 102, 183, 102, 157, 149, 150$$

2. Dans l'AVL obtenu ci-dessus, suppons qu'on supprime la clé 301 suivie de la suppression de la clé 342,
   (a) que devient l'arbre résultant ?
   (b) Est-ce encore un AVL ?
   (c) Sinon, appliquer la ou les rotations nécessaires pour lui rendre la propriété d'AVL.

ANNEXE

```
tas=structure
        arbre:tableau[1..tailleStock] d'objet;
        tailleTas:entier;
     finstructure;
curseur=entier;
sommet=entier;
```

Primitives de Tas :

```
fonction getValeur(ref T:tas d'objet;val s:sommet):objet;
fonction valeur(ref T:tas d'objet):objet;
fonction filsGauche(val s:sommet):sommet;
fonction filsDroit(val s:sommet):sommet;
fonction pere(val s:sommet):sommet;
fonction tasPlein(ref T:tas d'objet):booleen;
fonction setValeur(ref T:tas d'objet;val s:sommet;val x:objet):vide;
fonction créerTas(ref T:tas d'objet; val x:objet):vide;
```

```
type dico= structure
            a: sommet; /* l'arbre */
            p: sommet; /* le curseur */
           finstructure

arbreBinaire=curseur;
sommet=curseur;
```

Le type sommet présente les primitives suivantes :

- accès

```
        fonction getValeur(val S:sommet):objet;
        /* vaut NIL si le sommet n'existe pas /*
        fonction filsGauche(val S:sommet):sommet;
        fonction filsDroit(val S:sommet):sommet;
        fonction pere(val S:sommet):sommet;
```

- modification

```
        fonction setValeur(ref S:sommet;val x:objet):vide;
        /* affecte au sommet S la valeur x */
        fonction ajouterFilsGauche(ref S:sommet,val x:objet):vide;
        /* filsGauche(S)==NIL doit être vérifié */
        fonction ajouterFilsDroit(ref S:sommet,x:objet):vide;
        /* filsDroit(S)==NIL doit être vérifié */
        fonction supprimerFilsGauche(ref S:sommet):vide;
        /* filsGauche(S) est une feuille */
        fonction supprimerFilsDroit(ref S:sommet):vide;
        /* filsDroit(S) est une feuille */
        fonction detruireSommet(ref S:sommet):vide;
        /* S est une feuille */
```

– Il faut par ailleurs pouvoir créer la racine d'un arbre on a donc de plus la primitive

```
fonction creerArbreBinaire(val Racine:objet):sommet;
```

```
listeSC= liste de type_predefini;
```

défini en cours avec les primitives suivantes :

```
fonction valeur(val L:liste d'objet):objet;
fonction debutListe(ref L:liste d'objet);
fonction suivant(ref L:liste d'objet);
fonction listeVide(val L:liste d'objet): booleen;
fonction getCleListe(val L: liste d'objet):curseur;
fonction creerListe(ref L:liste d'objet):vide;
fonction insererApres(ref L:liste d'objet;
                      val x:objet;):vide;
fonction insererEnTete(ref L:liste d'objet
                       val x:objet):vide;
fonction supprimerApres(ref L:liste d'objet):vide;
fonction supprimerEnTete(ref L:liste d'objet):vide;
fonction setCleListe(ref L: liste d'objet;val c:curseur):vide;
fonction detruireListe(ref L:liste d'objet):vide;
```