Algorithmique 1 : Devoir Surveillé 3

Arbres binaires de recherche, tas et file de priorité et tables-hash

Durée : 60mn
Sans documents

Exercice 3.1 *Questions*

Cocher les affirmations qui sont correctes :

	Dans un arbre binaire de recherche le minimum est toujours à la racine.
	Le temps d'accès à l'élément maximum dans un tas min est $O(1)$.
	Pour un arbre binaire de recherche donné l'obtention de la liste des nombres triés est en temps $O(n)$.
	Dans un tas, la primitive `supprimerValeur` consiste à supprimer une valeur située dans une feuille.
	La complexité mémoire d'une table de hachage chaînée est plus importante que la complexité mémoire d'une table de hachage ouvert.
	Dans une table de hachage chaînée, l'exécution d'une recherche s'effectue en moyenne en $O(1)$.

Exercice 3.2 *Arbres binaires de recherche (ABR)*

1. Ecrire une fonction `tableauToABR` qui transforme un tableau d'entiers en ABR. La fonction doit renvoyer vrai si tous les éléments sont des entiers différents et faux sinon. On donnera l'algorithme complet d'insertion.
2. Appliquer cette fonction au tableau : `T={ 5, 8, 2, 6, 3, 4, 1, 7 }` et dessiner le résultat.
3. A quoi sert l'équilibrage d'un ABR ?
4. Dessiner l'ABR équilibré qui correspond à l'ABR obtenu précedemment.

Exercice 3.3 *Tas*

1. La séquence `{23, 17, 14, 6, 13, 10, 1, 5, 7, 12}` forme-t-elle un tas ? Justifier votre réponse.
2. Un tableau trié à rebours forme-t-il un tas ? Justifier votre réponse.
3. Soit le tableau `Tab={16, 14, 10, 8, 7, 9, 3, 2, 4, 1}`. Dessiner le tas max obtenu par ajouts successifs des éléments du tableau `Tab` en respectant l'ordre donné.
4. Soit un tas max, écrire la primitive
   `fonction supprimer(ref T: Tas de objet): vide`

Exercice 3.4 *Tables de hachage*

1. On considère l'insertion des clés `10, 22,31,4,15,28,17,88,59` dans une table de hachage de longueur $m = 9$ en utilisant l'adressage ouvert et un sondage linéaire : $s(k, i) = (h(k) + i)[m]$, k étant une clé, h la fonction de hachage principale, $h(k) = k[m]$, et $i = 0, \ldots, m - 1$.
   Dessiner la table de hachage résultat de l'insertion de cette suite de clés.
2. On considère la même suite de clés `10, 22,31,4,15,28,17,88,59` qu'on souhaite insérer dans une table de hachage de longueur $m = 9$ en utilisant l'adressage chaîné.
   Dessiner la table de hachage résultat de l'insertion de cette suite de clés.
3. Comparer les deux tables de hachage en termes de compléxité mémoire et temps de recherche d'une clé donnée.

ANNEXE A **Type abstrait** *arbreBinaire*

```
arbreBinaire= curseur;
sommet= curseur;
fonction creerArbreBinaire(val Racine:objet):sommet;
fonction detruireArbreBinaire(ref S:sommet):vide;
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;
```

ANNEXE B **Implémentation du type abstrait** *arbreBinaire*

```
cellule=structure
         info:objet;
         gauche: sommet;
         droit: sommet;
         pere: sommet;
       finstructure
sommet= ^cellule;
arbreBinaire= sommet;
```

ANNEXE C : **Arbres binaires de recherche** *ABR*
On utilise les primitives des arbres binaires. De plus, les primitives `ajouter` et `supprimer` permettent de faire évoluer un ABR.

```
fonction ajouter(ref x: sommet, val e: objet): vide;
fonction supprimer(ref x: sommet): booleen;
```

ANNEXE D : **Type abstrait tas.**

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

ANNEXE E : **Implémentation du type abstrait tas.**

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

```
sommet=entier;
fonction getValeur(val T: tas d'objet, val s: sommet): objet;
fonction valeur(val T: tas d'objet): objet;
fonction filsGauche(val s: sommet): sommet;
fonction filsDroit(val s: sommet): sommet;
fonction pere(val s: sommet): sommet;
fonction setValeur(ref T: tas d'objet, val s: sommet, val x:objet): vide;
fonction tasPlein(val T: tas d'objet): booleen;
fonction creerTas(ref T: tas d' objet, val racine: objet): vide;
fonction ajouter(ref T: tas d'objet, val v: objet): vide;
fonction supprimer(ref T: tas d'objet): vide;
```

Annexe F : **Type abstrait file de priorité.**
Primitive supplémentaire :

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

Annexe G : **Type abstrait tableHash**

```
fonction creerTableHachage(ref T: tableHash de cle, ref h: fonction): vide;
fonction charcher(ref T: tableHash de cle, val v: cle): curseur;
ajouter(ref T: tableHash de cle, val v: cle): booleen;
supprimer(ref T: tableHash de cle, val v: cle): vide;
```

Annexe H : **Adressage chaîné**

```
tableHash de cle= structure
                    table: tableau[0..m-1] de listeDC de cle;
                    h: fonction(val v: cle): curseur;
                  finstructure
```

Annexe I : **Adressage ouvert**

```
tableHash de cle= structure
                    table: tableau[0..m-1] de cle;
                    h: fonction(val v: cle): curseur;
                    s: fonction(val v: cle, i: entier): curseur;
                  finstructure
```