Université Bordeaux 1 LST Mention Informatique Semestre 3 2012-2013

Algorithmique 1 : Devoir Surveillé 2

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

Durée : 60mn
Sans documents

Exercice 2.1 *Arbres binaires de recherche (ABR)*
Soit l'arbre binaire de recherche illustré sur la Fig. 1(a)

1. Ecrire une fonction qui construit l'arbre de la Fig. 1(a). On supposera données toutes les primitives nécessaires.
2. On sait que la complexité des opérations sur les ABRs dépend de la hauteur de l'arbre. Ecrire une fonction `construire` qui à partir d'un ABR passé en paramètre construit un ABR équilibré aussi proche que possible d'un arbre binaire parfait de manière à ce que la hauteur soit minimum.
   Ainsi, la fonction `construire` appelée sur l'ABR de la Fig. 1(a) produira l'ABR de la Fig. 1(b).

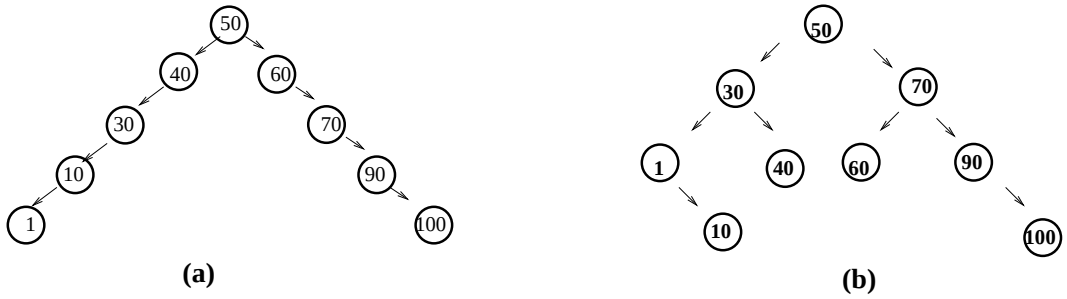

FIG. 1 – Equilibrage d'un ABR

Exercice 2.2 *Tas et files de priorité*

1. Soit la suite d'entiers `{ 1, 10, 30, 40, 50, 60, 70, 90, 1}`. Dessiner le tas obtenu par les ajouts successifs de ces entiers dans l'ordre indiqué.
2. On considère un ensemble d'avions. Chaque avion est identifié par sa destination représentée par un tableau de caractères, et sa priorité, donné par un entier.

```
avion= structure
  destination: tableau[1..10] de char;
  priorite: entier;
finstructure
```

   On représente un ensemble d'avions comme une file d'avions en utilisant comme clé la priorité. L'avion avec la priorité maximale est au sommet de la file.
   Une file d'avions est illustrée sur la Fig. 2.

– On change la priorité de l'avion `a={"bordeaux", 10}` en `a={"bordeaux", 15}`.
Dessiner la nouvelle file correspondante à ce changement.
– Ecrire la primitive
`fonction changeValeur(ref FA: file d'avion, val s: sommet, val nvellePri: entier):v`

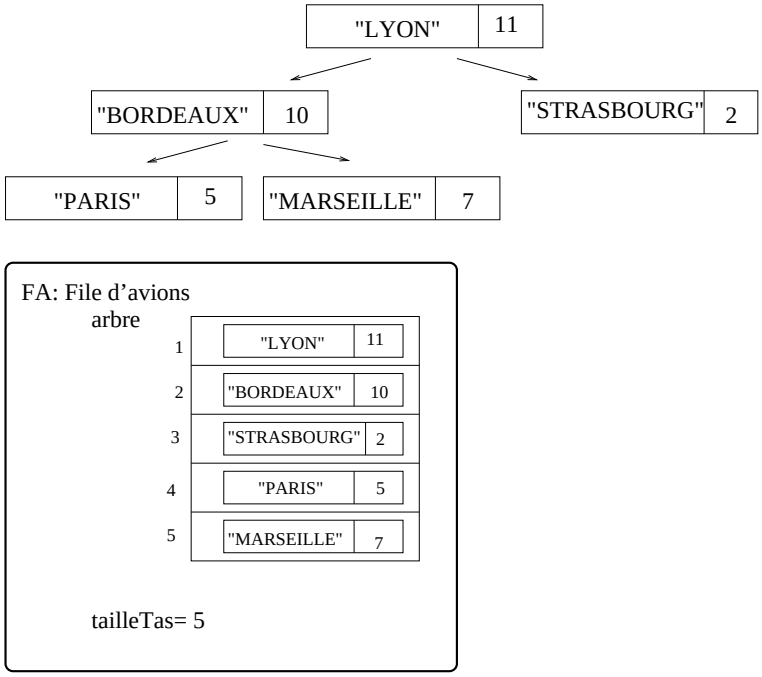

FIG. 2 – File d'avions

Exercice 2.3 *Tables de hachage, adressage ouvert*

1. Montrer comment on réalise l'insertion des clés `79, 69, 98, 72, 14, 50` dans une table à adressage ouvert en utilisant un double hachage :

$$h(k,i) = (h_1(k) + ih_2(k)) \text{ mod } \text{m}$$

$$h_1(k) = k \text{ mod } \text{m}$$

$$h_2(k) = 1 + (\text{ k } \text{ mod } (\text{m-2}))$$

On supposera un table de hachage de taille `m=13`.

2. Proposer une implémentation de cette table de hachage.

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
```