Algorithmique 1 : Devoir Surveillé 4

Dictionnaires et tables de hachage

Durée : 1h20mn
Sans documents.
Les primitives nécessaires à l'écriture des fonctions demandées sont données en annexe.

1 Dictionnaires

Exercice 4.1 *Insertion*
Dessiner le dictionnaire résultat de la suite d'opérations donnée :

```
var d: dico;
creerDictionnaire(d);
ajouter(d,"acos");
ajouter(d,"atoi");
ajouter(d,"char");
supprimer(d,"define" );
ajouter(d,"exp");
ajouter(d,"ceil");
supprimer(d,"cos");
ajouter("float");
ajouter("atol");
ajouter("floor");
ajouter("ctime");
supprimer("acos");
```

Exercice 4.2 *Mystere*
Quel est le résultat de la fonction `mystere` appliquée sur le dictionnaire construit dans l'exercice précédent ?

```
fonction mystere(var d: dico): entier
debut
  d.p= d.a;
  si (d.p == NIL) alors
    retourner 0;
  si (filsGauche(d.p) == NIL) alors
    retourner 1 + mystere(filsDroit(d.p));
  retourner mystere(filsGauche(d.p)) + mystere(filsDroit(d.p));
fin
```

2 Tables de hachage

Exercice 4.3 *Adressage direct*
On considère un ensemble dynamique `S` représenté par une table `T` à adressage direct et de longueur `m`. Décrire une fonction qui trouve l'élément maximum de `S`. Quelle est l'efficacité de votre fonction dans le pire des cas ?

Exercice 4.4 *Table de hachage*
Montrer comment on réalise l'insertion des clés `5, 28, 19, 15, 20, 33, 12, 17, 10` dans une table de hachage où les collisions sont résolues par chaînage.
On suppose que la table est de dimension `9`, et que la fonction de hachage est `h(k)= k mod 9`

Exercice 4.5 *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.
3. Ecrire la primitive `supprimer`.

Annexe A : Type abstrait containeur

```
valeur(val c: conteneur): objet;
creerConteneur(ref c: conteneur): vide;
ajouter(ref c: conteneur, x: objet): vide;
supprimer(ref c: conteneur, x: objet): vide;
detruireConteneur(ref c: conteneur): vide;
```

Annexe B : Type abstrait arbreBinaire.

```
arbreBinaire= curseur;
sommet= curseur;
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 creerArbreBinaire(val Racine:objet): sommet;
```

Annexe C : Type abstrait Dictionnaire.

```
fonction appartient(val d: dictionnaire, val M: mot): booleen;
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;
```

Annexe D : Implémentation du type abstrait Dictionnaire dans le type arbreBinaire.

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

Annexe E : 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 F : Adressage chaîné.

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

Annexe G : 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
```