

Algorithmes complexes Partie 2

Les tables

Définition

- Une table est une structure de données telle que l'accès à un élément est déterminé à partir de sa clé.
- Une clé permet d'identifier un élément de manière unique : la clé est dite discriminante.
- L'allocation d'un table est généralement contiguë.
- La partie **infos** contient les informations spécifiques à l'application.

	Clé	Infos
0		
1		
2	Dupond	...
3		
N-1		

Index

- Si une structure données (Infos) contient plusieurs éléments discriminants, il est possible alors de créer une table servant d'index et utilisant comme clé l'élément discriminant choisi.
- Un index permet d'accélérer les traitements de recherche.
- Exemple :

Index sur Nom	
Nom	Pos
Henri	0
Dupond	25

Index sur Numéro	
Numéro	Pos
0630302456	0
0630302456	25

Tableau de données				
Pos	Nom	Prénom	Adresse	Numéro
0	Henri	Arthur	Rue là	0478923210
	...			
25	Dupond	Michel	Rue Ici	0630302456

Rappel sur les clés

- Sorte Clé
Utilise Booléen
- Opérations :
- \leq : Clé \otimes Clé \rightarrow Booléen ;
 - $=$: Clé \otimes Clé \rightarrow Booléen ;

Avec :
 x, y, z : Clé

- Axiomes :
 $x \leq y = \text{vrai}$
 $(x \leq y) \wedge (y \leq x) \Rightarrow x = y$
 $(x \leq y) \wedge (y \leq z) \Rightarrow x \leq z$

On appelle clé l'application, qui à chaque éléments associe sa clé :
Clé: Élément \rightarrow Clé

Spécification du type

- Sorte Table
Utilise Élément, Clé, Entier, Booléen
- Opérations:

Créer-Table	: Entier → Table
Insérer	: Table ⊗ Élément → Table
Rechercher	: Table ⊗ Clé → Élément
∈	: Élément ⊗ Table → Booléen
Supprimer	: Table ⊗ Clé → Table
TailleMax	: Table → Entier
Taille	: Table → Entier
Nième	: Table ⊗ Entier → Élément
- Avec
 - I : entier;
 - T : Table;
 - E : Élément;
- Pré-condition :
 - Rechercher (T, Clé(E)) est défini si et seulement si E ∈ T

Spécification du type

- Axiomes
 - Taille(Créer-Table(i)) = 0
 - TailleMax(Créer-Table(i)) = i
 - $I < 0 \Rightarrow E \in \text{Insérer}(\text{Créer-Table}(i), E)$
 - $I < 0 \Rightarrow \text{Rechercher}(\text{Insérer}(\text{Créer-Table}(i), E), \text{Clé}(E)) = E$
 - $E \in T \Rightarrow \text{Taille}(\text{Insérer}(T, E)) = \text{Taille}(T)$
 - $E \in T \Rightarrow E \in \text{Insérer}(T, E)$
 - $E \notin T \wedge \text{Taille}(T) < \text{TailleMax}(T) \Rightarrow \text{Taille}(\text{Insérer}(T, E)) = \text{Taille}(T) + 1$
 - $E \notin T \wedge \text{Taille}(T) < \text{TailleMax}(T) \Rightarrow E \in \text{Insérer}(T, E)$
 - $E \notin T \wedge \text{Taille}(T) < \text{TailleMax}(T) \Rightarrow \text{Rechercher}(\text{Insérer}(T, E), \text{Clé}(E)) = E$
 - $\text{TailleMax}(T) = \text{TailleMax}(\text{Insérer}(T, E))$

Spécification du type

- $E \in T \Rightarrow \text{Taille}(\text{Supprimer}(T, \text{Clé}(E))) = \text{Taille}(T) - 1$
- $E \in T \Rightarrow E \notin (\text{Supprimer}(T, \text{Clé}(E)))$
- $E \notin T \Rightarrow \text{Taille}(\text{Supprimer}(T, \text{Clé}(E))) = \text{Taille}(T)$
- $E \notin T \Rightarrow \text{Supprimer}(T, \text{Clé}(E)) = T$
- $E \notin T \Rightarrow E \notin (\text{Supprimer}(T, \text{Clé}(E)))$
- $E, E2 \in T \wedge E \neq E2 \Rightarrow \text{Rechercher}(\text{Supprimer}(T, \text{Clé}(E)), \text{Clé}(E2)) = E2$
- $\text{TailleMax}(T) = \text{TailleMax}(\text{Supprimer}(T, E))$
- $j \geq 0 \wedge j < i \Rightarrow \text{Nième}(\text{Insérer}(\text{Créer-Table}(i), E), j) = E$

Recherche dichotomique

- La recherche dichotomique n'est possible que sur un ensemble trié
- Principe de base : au lieu de parcourir séquentiellement la structure, on la parcourt en prenant le milieu de l'espace de recherche:
 - Si on a trouvé l'élément recherché, l'algorithme s'arrête en renvoyant l'index.
 - Sinon on prend le milieu de l'espace restant, en choisissant la moitié gauche ou la moitié droite, suivant si l'élément recherché est inférieur ou supérieur à l'élément courant observé.

Recherche dichotomique

Fonction Rechercher(T : Table ; C : Clé) : Element
debut, fin, i : Entier;
trouve : Booléen;

Début

```
debut := 0;  
fin := TailleMax(T) - 1;  
trouve := faux;  
TANTQUE (debut <= fin) ET (!trouve) FAIRE  
  i := (debut+fin) / 2;  
  SI Clé(Nième(T, i)) = C ALORS trouve := vrai  
  SINON SI (Clé(Nième(T, i)) > C) ALORS fin := i - 1  
  SINON debut := i + 1
```

FINSI

FINSI

FINTANTQUE

retourne Nième(T, i);

Fin

ALGORITHME DE HACHAGE

- Structures ordonnées : la place d'un élément dépend de la valeur de sa clé comparée à celle des autres éléments.
- Différence avec les tables : la clé n'est plus discriminante
- Technique de hachage : la place d'un élément n'est calculée qu'à partir de sa clé.
- Calcul de la place réalisé à l'aide de la fonction de hachage.
- Calcul à partir de la clé d'une adresse mémoire ou d'un indice de tableau.
- Opérations de recherche, d'adjonction et de suppression en temps moyen constant.
- Le temps d'accès ne dépend plus du nombre d'élément dans la structure.

Exemple

- Soit E un ensemble de prénoms = { serge, odile, luc, anne, annie, jean, julie, basile, marcel, élise}
- On associe à chaque élément e de E un nombre h (e) compris entre 0 et 12
 1. Attribuer aux lettres a, b, c, ..., z les valeurs 1, 2, ..., 26.
 2. Ajouter les valeurs de lettres de e.
 3. Ajouter au nombre obtenu, le nombre de lettre de e.
 4. Calculer ce nombre modulo 13 pour obtenir le nombre h (e) recherché

Exemple (Suite)

- D'où les résultats suivant :
h (serge) = (54 + 5) mod 13 = 7
h (odile) = (45 + 5) mod 13 = 11
h (luc) = (36 + 3) mod 13 = 0
...
h (élise) = 3
- h est injective : on peut ranger chaque élément e à l'indice que l'on vient de calculer
- h non injective : $x_1 \neq x_2$ et $h(x_1) = h(x_2)$: collision primaire:
 - Il n'est pas possible dans le cas général d'éviter des collisions.
 - La fonction de hachage ne donne pas accès à un élément mais à un petit ensemble d'éléments en petits morceaux.

Pos	clé	Élément
0	0	Luc
1	1	
2	2	Basile
3	3	Élise
4	4	Paula
5	5	
6	6	Marcel
7	7	Serge
8	8	jean
9	9	Annie
10	10	Julie
11	11	Odile
12	12	Anne

Principes généraux du hachage

- Gestion d'une collection C d'éléments, les clé appartiennent à univers U très grand.
 - Taille de C relativement petite par rapport au nombre d'éléments de U
 - **Exemple :**
C est un ensemble de 1000 mots de 10 lettres
U est l'ensemble de tous les mots possibles de 10 lettres
 $\text{card}(U) = 10^{10}$
- On ne peut réserver une place mémoire pour chacun des éléments de U
On utilise une fonction de hachage h :
 $h : U \rightarrow [1..m]$
où l'entier m est choisi en fonction de la taille de C
- ☞ La fonction de hachage h n'est pas injective.
- x, clé, h(x) indice de la place de x dans un tableau t de n éléments :
h(x) est la valeur de hachage primaire.
 - h(x) sert à vérifier si x appartient à t, à ajouter ou à le supprimer.
 - On doit choisir la fonction de hachage de façon à obtenir une répartition aussi uniforme que possible des éléments.

Propriétés de la fonction de hachage

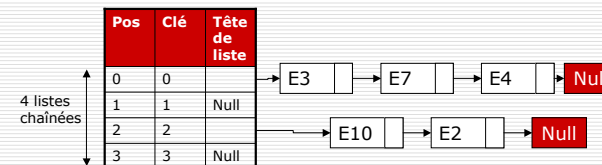
- Calcul de la fonction de hachage aussi rapide que possible.
- La fonction doit être uniforme afin de limiter les collisions :
 $\forall x \in U \text{ et } \forall i \in [1..m], \text{ probabilité } (h(x) = i) = (1/m)$
- Il faut gérer les inévitables collisions ; deux grands types de techniques :
 - Les méthodes de résolution des collisions par chaînage.
 - Les méthodes de résolution des collisions par calcul.

Résolution des collisions par méthodes indirectes

- On réalise un chaînage entre les éléments en collision.
- Localisation du chaînage :
 - Dans la zone de débordement à l'extérieur du tableau de hachage (méthode de hachage avec chaînage séparé).
 - A l'intérieur même du tableau (méthode de hachage coalescent) .

Hachage avec chaînage séparé

- Éléments en collision chaînés entre eux à l'intérieur du tableau de chaînage.
- Tableau constitué de m têtes de listes chaînées représentant chacune un ensemble d'éléments qui rendent un même valeur par la fonction hachage.
- Algorithme de recherche, d'adjonction et de suppression sont très proches de ceux des listes chaînées.
- Complexité fonction de la longueur de ces listes (aussi courtes que possible).



Hachage coalescent

- Le hachage coalescent peut se programmer :
 - Si possibilité d'allocation dynamique de mémoire.
 - En réservant à l'avance une zone contiguë de mémoire.
- Tableau de taille m , on réserve à priori une zone d'adresse primaire de capacité p et une réserve de capacité r pour gérer les collisions :
 - $m = p + r$
 - La fonction de hachage est à valeur dans $[1..p]$.

	Pos	Clé	Éléme nt
Zone d'adresses primaires	0	0	E3
	1	1	Vide
	2	2	E10
Réserve	3	3	Vide
	4	0	E7
	5	0	E4
	6	2	E2

Algorithmes complexes : Les
Tables

17

Hachage coalescent (suite)

- Difficulté du choix de bonne proportion :
 - Réserve trop petite : saturation avant utilisation suffisante du tableau.
 - Réserve trop grande : diminution de l'effet de dispersion.
- Pas de distinction entre les zones du tableau :
 - Toute place peut être atteinte par le hachage primaire.
 - Toute place peut servir à résoudre une collision.
 - Création de collisions secondaires

Algorithmes complexes : Les
Tables

18

Hachage coalescent : Exemple sans distinction entre les zones du tableau

- $h(x_1) = h(x_2) = k$
 - on met x_2 à la fin du tableau (indice m)
 - $h(x_3) = m$, il y a collision secondaire.
- 1^{er} collision : on place l'élément à la fin du tableau.
- 2^{ème} collision : on ne peut placer x_3 à sa vrai place : on le met j et on chaîne j avec m .
- ☞ Collisions secondaires : temps de recherche plus long qu'avec chaînage à l'extérieur du tableau.

Pos	Clé	Éléme nt
...
k	k	X1
j		
m		

Pos	Clé	Éléme nt
...
k	k	X1
m	k	X2

Pos	Clé	Éléme nt
...
k	k	X1
j	m	X3
m	k	X2

Algorithmes complexes : Les
Tables

19

Addition avec un hachage coalescent et une zone de réserve

- Algorithme d'addition

```
Type Tab : Tableau [ 1..n ] de Article
Enregistrement Article {
    val : élément ;
    lien : 0..m ;
}
```

Soit une fonction indiquant si une place est ou non occupée.
Fonction Vide (t : Tableau ; i : 1..n) : Booléen ;

Algorithmes complexes : Les
Tables

20

Addition avec un hachage coalescent et une zone de réserve (suite)

```

Procédure Ajouter-hco ( x : élément ; In_Out t : Tableau ; In_Out plein : Booléen ) ;
R, J : Entier ;
Début
  i := h ( x ) ; R := m ; plein := faux ;
  SI Vide( t , i )
  ALORS t [ i ] . val := x ; t [ i ] . lien := -1 ;
  SINON
    TANTQUE ( t [ i ] . val ≠ x ) ET ( t [ i ] . lien ≠ -1 ) FAIRE
      i := t [ i ] . lien ;
    FINTANTQUE ;
    SI t [ i ] . val ≠ x ALORS
      TANTQUE ( R ≥ 1 ) ET Vide ( t , R ) = faux FAIRE
        TANTQUE
          R := R - 1 ;
        FINTANTQUE ;
        SI R ≥ 1 ALORS
          t [ i ] . lien := R ;
          t [ R ] . val := x ;
          t [ R ] . lien := -1 ;
        SINON
          plein := vrai ;
        FINSI ;
      FINSI ;
    FINSI ;
  FINSI ;
Fin
  
```

Résolution des collisions par calcul (Méthodes directes)

- Principes :
 - Pas de chaînage : on utilise l'espace pour diminuer le nombre de collisions.
 - Méthodes de résolution de collisions par calcul à l'intérieur du tableau.
 - Définition d'une fonction des essais successifs :
Essai : $U \rightarrow \{ 1, 2, 3, \dots, m \}$
 - $x \in U$, Essai (x) : Essai₁ (x) , Essai₂ (x) , ..., Essai_m (x) est une permutation de $\{ 1, 2, \dots, m \}$
 - $\forall i , \text{Essai}_i (x) \in \{ 1, \dots, m \}$
 - $i \neq j \Rightarrow \text{Essai}_i (x) \neq \text{Essai}_j (x)$
 - Lors d'une recherche de x dans le tableau t [1..m], on explore successivement les places Essai₁ (x) (valeur de hachage primaire) puis Essai₂ (x) , Essai₃ (x)....

Résolution des collisions par calcul (Méthodes directes) et algorithme de recherche

- Plusieurs cas possibles :
 - On trouve x
 - On arrive sur une place vide
 - On arrive à la fin des m essais
- Algorithme de recherche
 - Fonction Essai (i : 1..m ; x : élément) : 1..m ;
 - Recherche dans un simple tableau d'éléments (et non plus d'articles)
 - Type Tab = Tableau [1..m] de élément ;
 - Fonction indiquant si une case du tableau est occupée ou pas.
 - **Fonction** Vide (t : Tab ; i : 1..m) : Booléen ;

algorithme de recherche

```

Fonction Rechercher-HD ( n : élément ; t : Tab ) : 0..m ;
i, v : Entier ;
Sortie : Booléen ;
Résultat : Entier ;
Début
  i := 1 ; Sortie := faux ;
  TANTQUE ( i < m ) ET ¬ Sortie FAIRE
    v := essai ( x , i ) ;
    SI ( Vide ( t , v ) OU t [ v ] = x )
      ALORS Sortie := vrai ;
    FINSI ;
    i := i + 1 ;
  FINTANTQUE ;
  SI t [ v ] = x
    ALORS Résultat := v
  SINON Résultat := 0
  FINSI ;
  Retourne Résultat ;
Fin
  
```

algorithme d'ajout

- On place x à la première place libre rencontrée lors des essais successifs.

```
Procédure Ajouter-HD ( x : élément ; In_Out t : Tab ; In_Out plein : Booléen ) ;
Var
  i , v : 1..m ;
  Sortie : Booléen ;
Début
  i := 1 ; plein := faux ; Sortie := faux ;
  TANTQUE ( i ≤ m ET ¬ Sortie ) FAIRE
    v := essai ( i , x ) ;
    SI ( Vide ( t , v ) OU t [ v ] = n )
      ALORS Sortie := vrai ;
    FINSI ;
  FINTANTQUE ;
  SI t [ v ] ≠ x
    ALORS
      SI Vide ( t , v )
        ALORS t [ v ] := x ;
        SINON plein := vrai ;
      FINSI ;
    FINSI ;
Fin
```

Exemples de méthodes directes : Hachage linéaire

- En cas de collisions sur l'indice v , on essaie l'indice $v + 1$
- Si on est à la fin du tableau ($v = m$), on recommence au début du tableau
- \oplus opération assurant cette progression
 - $\oplus : [1..m] \oplus [1..m] \rightarrow [1..m]$
 - $(a , b) \rightarrow a + b$ si $a + b \leq m$
 - $(a , b) \rightarrow a + b - m$ si $a + b > m$
- On fait la suite d'essais suivante :
 - $\text{essai}_1 (x) = h (x)$
 - $\text{essai}_2 (x) = h (x) \oplus 1$
 -
 - $\text{essai}_i (x) = h (x) \oplus i - 1$
 -
 - $\text{essai}_m (x) = h (x) \oplus m - 1$
- Inconvénient : formation de groupements d'éléments contigus. Nécessite une bonne dispersion initiale des éléments.

Exemples de méthodes directes : Double hachage

- 2 fonctions servent à calculer la valeur de l'indice
 - $D(x)$: Fonction de décalage
 - $\text{essai} (x) = h (x) \oplus d (x) \cdot (i - 1)$
- Si $d (x)$ constant : répartition régulière de k en k . On cherche au contraire à les "dispenser"
- Il faut bien engendrer une permutation de $[1..m]$: $d (x)$ premier avec m pour tout x .
- Deux solutions :
 - m premier, d à valeurs dans $[1..m - 1]$
 - $m = 2p$ et $d (x)$ impair pour tout x , par exemple : $d (x) = 2 \cdot d' (x) + 1$ et d' à valeurs dans $[0 , 2^{p-1} - 1]$