Structures de Données
Les tables de hachage

Licence IUP-MIAGE

2003-2004

Pascal PETIT (sur la base d'un support de Guillaume HUTZLER (2003-2004))

Le hachage (1)

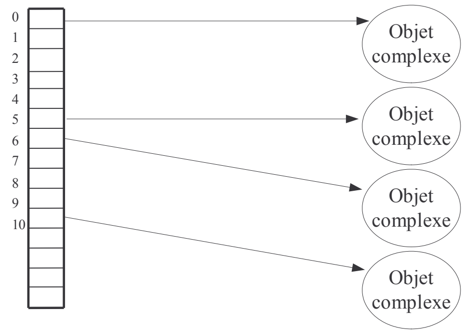
- Un dictionnaire: insertion, suppression et accès à un élément en fonction de sa clef
- Dans les SD vues en cours: les éléments sont placés dans la structure en fonction de la valeur de leur clefs par rapport à celle des autres éléments.
  - La place dépend des autes éléments
  - L'accès n'est pas instantannée car nécessite une recherche

Le hachage (2)

- Dans un tableau: accès direct si on connait l'indice de l'élément
- Structure efficace pour les opérations:
  - Ajout d'un élément ayant un indice donné
  - Recherche d'un élément ayant un indice donné
  - Suppression d'un élément ayant un indice donné
- Dictionnaire avec les indices du tableau comme clef.
- Utilisé pour la représentation des ensembles sous la forme de tableaux de booléens

•Le hachage (2) : représensation

- En pratique; tableaux de pointeurs vers des objets complexes; NULL si pas d'objet:



•Le Hachage (3) :

- Problèmes
  - Il faut autant d'indices dans le tableau que de valeurs possibles pour la clef choisie
  - Perte importante d'espace (exemple: 5 éléments ayant comme clef 2,10, 100, 5000 et 10000 -> tableau à 10001 entrées au moins)
- Solution utopique:
  - Associer à chaque clef un nombre entier appartenant à [0, n-1]s'il y a n éléments;
  - deux clefs différentes => deux entiers différents;
  - Pb: la détermination de l'indice équivaut à une recherche (contrainte trop forte)

Hachage (4)

12	1		15	20	9	2	7

•Quelque soit sa valeur, le prochain élément devra aller dans la troisième cellule

•La place ne peut donc plus être de la forme h(e)

•L'accès à un élément nécessite une recherche

•Le hachage (5): solution au problème

- L'indice doit être calculé uniquement en fonction de la clef indépendamment des valeurs des autres éléments : h(clef) = indice dans le tableau
- Ensemble des indices plus petit que l'ensemble des clefs => certaines clefs auront le même indice: Collision
- La complexité dépendra du calcul de h et de l'influence des collisions

•Hachage: Exemple concret :

- on a un ensemble d'une centaine de personnes dont on souhaite stocker le dossier dans une armoire
- indice: jour de naissance (sans l'année)
  - 366 indices possible pour 100 personnes => place perdues (mais pas trop)
  - Si plus de 23 personnes: plus de 50% de chance d'avoir une date commune => collision

Principe

- Supposons que
  - on souhaite représenter un ensemble E
  - on dispose d'une fonction
    `h : clé → [1..m]`
- Si, pour tout c et c' ∈ E, on a h(c) ≠ h(c'), alors
  - h(c) peut être utilisé comme indice d'un tableau T dont les bornes sont 1 et m
  - pour tout élément e, T[h(la_clé(e))] contient la représentation de e

Application pratique

- il est rare que l'on obtienne une fonction injective
- généralement, il existe deux clés c et c' pour lesquelles h(c) = h(c')
  - on parle de *collision*
  - *la fonction h permet alors de localiser, non pas un élément de E, mais une petite collection d'éléments de E représentée par une liste*
  - l'ensemble E est ainsi « haché en petits morceaux »

•*Gestion des collisions*

- *Par chaînage*
  - Chaque cellule d'indice i du tableau contient la liste des éléments e tels que h(e)=i
- Adressage ouvert: dans la table
  - h devient h(e,i) et on essaie successivement i=0, i=1, ... jusqu'à trouver une cellule vide.
  - La suppression est délicate (ne pas casser une chaîne existante he,0), h(e,1), ... h(e,i) en supprimant un élément au milieu (=> valeur spécifique indiquant la suppression)
- Dans la suite du cours, on ne traite que la gestion des collisions par chaînage

Critères pour le choix de `h`

- Le choix de h est primordial
  - il faudrait que toutes les valeurs entre 1 et m soient équiprobables
  - pour toute clé c et pour tout i entre 1 et m, la probabilité que h(c) ait pour valeur i doit être 1/m
- En pratique
  - la probabilité qu'il n'y ait pas de conflit pour un ensemble E de n éléments (n<m) est assez faible
    - exemple : dans un groupe de 23 personnes, 1 chance sur deux pour que deux personnes soient nées le même jour de l'année
  - il y a en fait très peu de collisions pour une valeur de i donnée

Critères pour le choix de `h`

- la fonction h doit être adaptée à la probabilité d'avoir une clé donnée
  - donc adaptée aux ensembles effectivement représentés
- elle doit être déterministe
  - renvoie toujours la même valeur pour une clé donnée
- elle doit être facile à calculer
  - bonnes performances en temps de calcul

Exemples de fonctions `h` (1)

- **extraction de bits**
  - clé représentée par une suite de bits, dont on extrait p bits
  - nombre entre 0 et 2^p-1
  - facile à mettre en œuvre
  - bons résultats que si les bits non pris en compte ne sont pas significatifs

Exemples de fonctions `h` (2)

- **compression de bits**
  - clé découpée en q tranches de p bits combinées par une addition ou un « ou exclusif »
    - si c = t[1]t[2]...t[q]
    - alors h(c) = t[1] xor t[2] xor ... xor t[q]
  - parfois décalage circulaire sur les tranches avant de les combiner
  - facile à mettre en œuvre
  - utilisée pour comprimer des clés très longues, en association avec une autre méthode

Exemples de fonctions `h` (3)

- **Division**
  - on prend le reste de la division par m de la représentation de la clé
  - `h(c) = c mod m`
- **Multiplication**
  - soit r, un réel tel que 0 < r < 1
  - `h(c) =` $\lfloor$`((c*r)mod1)*m`$\rfloor$
  - problème d'accumulation aux extrémités du tableau si r trop proche de 0 ou 1
  - marche bien avec $(\sqrt{5}-1)/2$ et $1-(\sqrt{5}-1)/2$

Recherche

Etant donné un élément e de clé c
  - calcul de l'indice dans le tableau et récupération de la liste d'éléments
  - recherche de l'élément dans cette liste

Ajout

- Etant donné un élément e de clé c
  - calcul de l'indice dans le tableau et récupération de la liste d'éléments
    - Si on est sur que l'élément n'y est pas : ajout en début de liste
    - Sinon : recherche de l'élément dans cette liste et si l'élément n'existe pas, ajout en fin de liste

Suppression

- Etant donné un élément e de clé c
  - calcul de l'indice dans le tableau et récupération de la liste d'éléments
  - recherche de l'élément dans cette liste
  - si l'élément existe, suppression de la liste

Complexité

- Si fonction correctement choisie
  - les listes ont toutes a peu près la même longueur
  - la longueur moyenne des listes est $\mu = n/m$
- complexité moyenne des opérations de recherche, d'ajout et de suppression proportionnelle à cette longueur moyenne
- d'autant plus faible que m est grand
  - mais entraîne un encombrement mémoire important