Les Tables de Hachage

NICOD JEAN-MARC

Licence 3 Informatique
Université de Franche-Comté
UFR des Sciences et Techniques

septembre 2007

Référence

- Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein. *Introduction à l'algorithmique*, 2^{e} édition, Dunod, 2002.

Sommaire

Généralités

- Opérations sur les ensembles de clés dynamiques :
  - table des symboles : `insérer`, `rechercher`, `supprimer` (les clés sont des chaînes de caractères)
  - Difficulté d'effectuer des accès directs
    - pas une case pour chaque clé
- Mise en œuvre efficace des dictionnaires
  - $O(n)$ dans le pire des cas et $O(1)$ en moyenne
- Il s'agit d'une généralisation des tableaux classiques
- Ici les données à stocker ont des clés qui permettent de calculer un `indice` dans la table et non des clés qui sont les `indices`

- Gestion des collisions : $\neq$ clés k $\rightarrow$ un même indice `i`

Sommaire

Tables à adressage direct
cas idéal théorique

définitions

- Technique simple si l'univers des clés U est petit

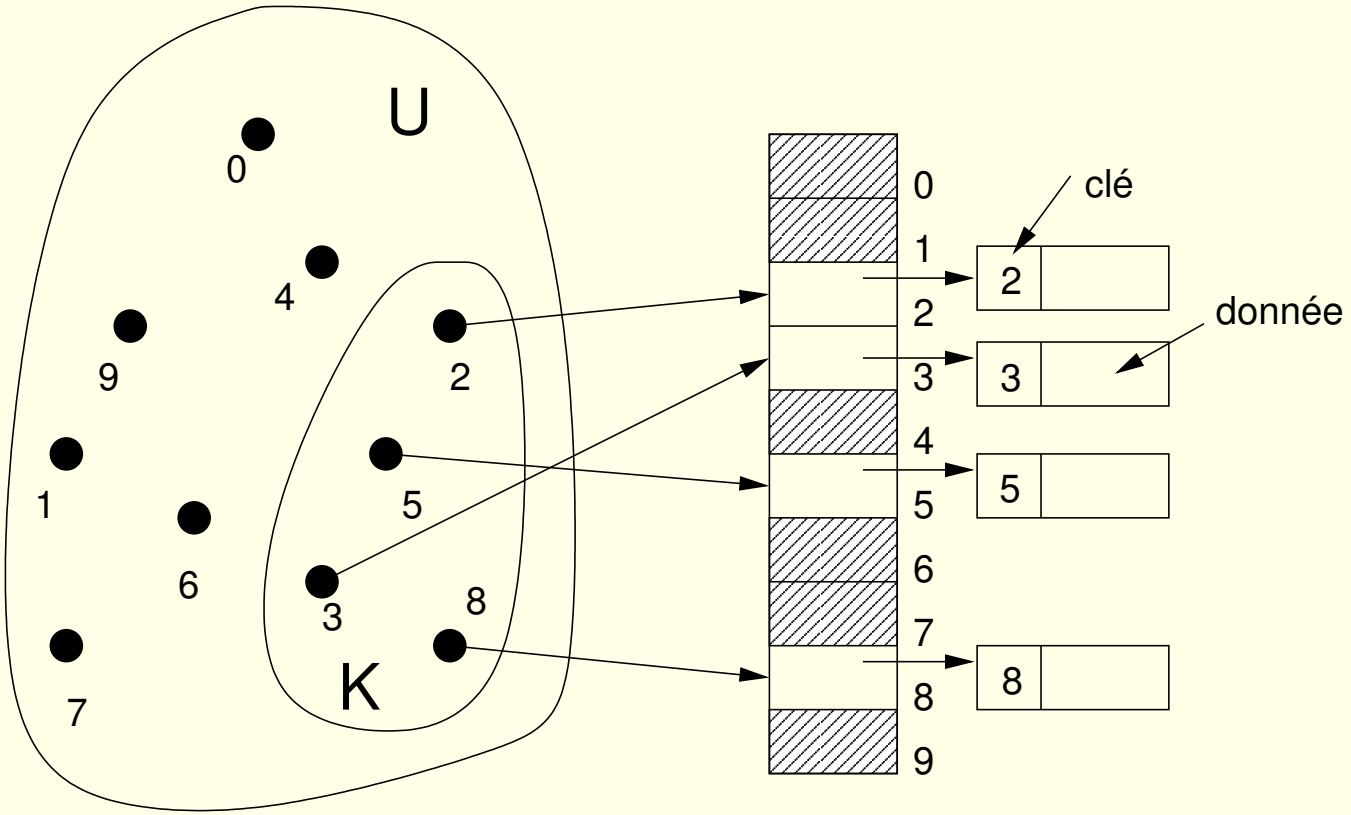
Soit $U = \{0, 1, \ldots, \mathtt{m} - 1\}$ avec $\mathtt{m}$ petit

- 2 éléments ne peuvent avoir la même clé

Soit K un sous ensemble dynamique dans lequel chaque élément possède une clé dans U

- Représentation de cet ensemble par une table à adressage directe $\mathtt{T}[0 \ldots \mathtt{m} - 1]$
- Chaque indice de $\mathtt{T}$ correspond à une clé dans l'univers U
- Impossible en pratique

Exemple d'adressage direct



Opérations en adressage direct

- `rechercherAdressageDirect(T,k)`
  - retour `T[k]`
- `insérerAdressageDirect(T,k)`
  - `T[clé[x]]` ← `x`
- `supprimerAdressageDirect(T,x)`
  - `T[clé[x]]` ← `null`

- Chaque opération est en O(1)
- Modif : stockage de l'élément directement dans `T` si la clé n'est pas toujours indispensable dans l'objet `x`

Sommaire

Les Tables de Hachage
cas pratique

définitions

Si U est grand, T ne tient pas en mémoire

- ☺ limitation du gaspillage de la mémoire
- ☺ si $|K| \ll |U|$ une table de hachage occupe moins de place
- ☹ l'accès en $O(1)$ seulement en moyenne alors que $O(1)$ dans le pire cas en adressage direct

Où stocke-t-on la clé k ?

- adressage direct : T[k]
- table de hachage : T[h(k)] avec h une fonction de *hachage*

Fonction de hachage

définition

Soit h une fonction de hachage :

$$h : U \rightarrow \{0, 1, \ldots, m-1\}$$

h(k) est la valeur de hachage de la clé k

- h établit une correspondance entre U et les cases de T, la table de hachage
- le but de la fonction de hachage est de réduire l'intervalle des indices du tableau à gérer
  - on gère m valeurs au lieu de $|U|$

Gestion des collisions

les collisions

Comme $|U| \gg m$ il y aura des collisions, c'est à dire :

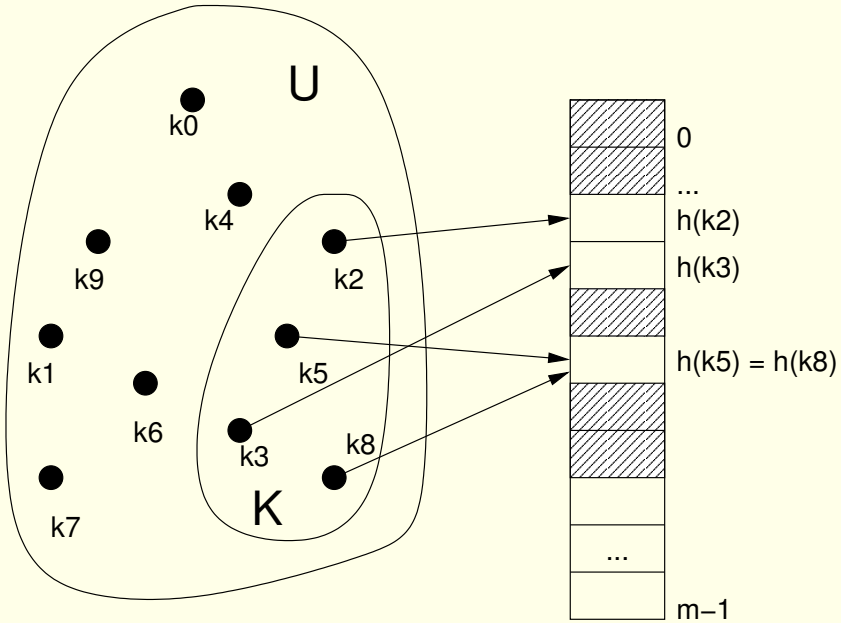
$$\exists\, k \text{ et } k' \in U \mid h(k) = h(k') \text{ avec } k \neq k'$$

L'idée est de trouver une fonction h qui semble aléatoire afin de limiter le nombre de collisions

Attention : h est une fonction déterministe sinon on ne pourrait pas retrouver nos données

- C'est pourquoi on parle de fonction de hachage

Exemple d'une collision



- Même s'il est possible de limiter le nombre de collisions, elles sont inévitables par construction
- Il existe des techniques pour gérer ces collisions

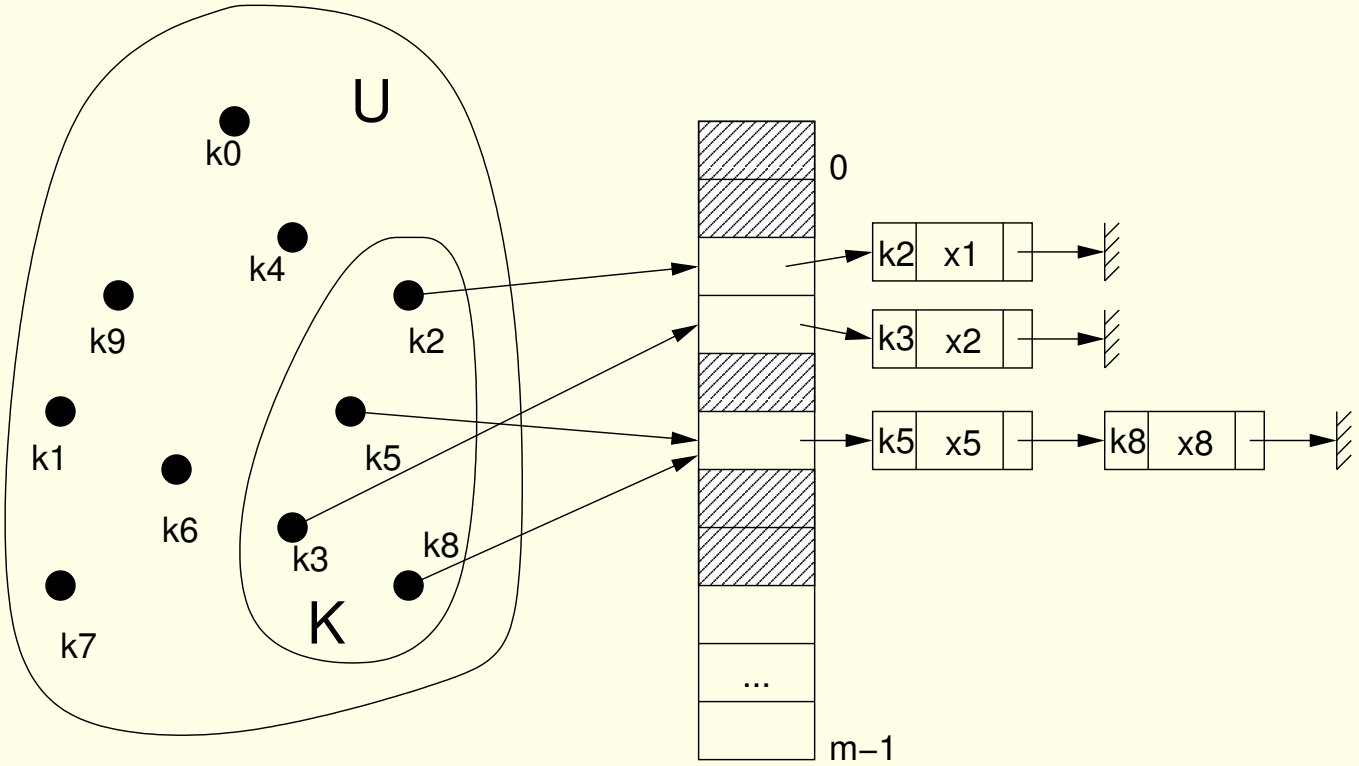
Résolution des collisions par chaînage

solution par chaînage

- Tous les éléments hachés vers la même case sont placés dans une liste chaînée
- La case `T[i]` contient le pointeur vers la liste des éléments de clés k tels que `h(k)=i`
- Si `T[i]` ne désigne rien, alors il pointe sur `null`

Exemple

mise en œuvre du chaînage sur l'exemple précédent



Opérations avec le chaînage

Les opérations sur `T` sont faciles à mettre en œuvre avec le chaînage :

- `insérerHachageChaîné(T,x)` :
  - insertion de `x` en tête de la liste `T[h(clé(x))]`
- `rechercherHachageChaîné(T,k)` :
  - recherche de l'élément de clé `k` dans `T[h(k)]`
- `supprimerHachageChaîné(T,x)` :
  - supprime `x` de la liste `T[h(clé(x))]`

Voir le cours sur les listes chaînées pour connaître la complexité de ces opérations.

- Comportement très mauvais dans le pire cas.
- Sa performance dépend de la performance de h

Sommaire

Définition

une bonne fonction de hachage ou hachage uniforme simple

- But : chaque clé a autant de chance d'être hachée vers l'une quelconque des m cases de T (impossible à vérifier en pratique)
- Condition d'existence : les clés k sont issues d'une distribution uniforme aléatoire

exemple : $0 \leq k \leq 1$ avec $h(k) = \lfloor km \rfloor$

- En pratique on recherche une fonction de hachage permettant, par exemple, à 2 clés ayant des motifs très voisins d'avoir des valeurs de hachage très différentes

Définition (suite)

clés et fonction de hachage

- Les clés sont des entiers, le plus souvent
- Sinon trouver un moyen de les voir comme tel
- Exemple :
  - une chaîne de caractères peut être convertie dans une certaine base (ici, base 128) :
  - la chaîne “pt” (caractères ascii 112 et 116) donne 14452 en base 128 : $112 \times 128 + 116 = 14452$

Dans la suite du cours, on suppose que les clés sont des entiers naturels $\mathbb{N}$

Méthode de la division

principe

$$h(k) = k \, mod \, m$$

- Méthode très rapide
- 2 clés proches peuvent avoir des clés très différentes
- Attention à la valeur choisie pour m :
  - ☹ $m = 2^p$ car $h(k)$ serait les p bits de poids faible de k. Tous les bits de la clé ne sont pas utilisés
  - ☹ $m \neq 2^p - 1$ si k est une chaîne de caractères en base 2^p car une permutation des caractères ne modifie pas $h(k)$
  - ☺ m est un nombre premier éloigné d'une puissance de 2, exemple :
    - 2000 chaînes de caractères 8 bits avec $m = 701$
    - 3 échecs en moyenne, ce qui est un bon rapport

Méthode de la multiplication

principe

$$h(k) = \lfloor m(kA \bmod 1) \rfloor \text{ avec } 0 < A < 1$$

$$kA \bmod 1 = kA - \lfloor kA \rfloor \text{ (partie décimale de } kA)$$

- ici m n'est plus une valeur critique
  - m est le plus souvent une puissance de 2
  - fonctionne mieux avec certaines valeurs de A comme le nombre d'or (*cf [Knuth 73] : "The art of Computer Programming"*): $A \simeq \frac{\sqrt{5}-1}{2}$

Hachage universel

mise en échec d'une fonction de hachage figée

- Il est possible de trouver n clés hachées vers la même case de T
  - acte de malveillance ou mauvais choix des identifiants pour une table des symboles (?)
  - temps de recherche en $O(n)$ en moyenne
  - choix aléatoire d'une fonction de hachage pour sortir du cas défavorable
- Cela définit le hachage universel
  - bonnes performances en moyenne à cause du caractère aléatoire du choix de la fonction de hachage

Hachage universel (suite)

collection finie de fonctions de hachage

- Soit $\mathcal{H}$ une collection finie de fonctions de hachage

$$h : U \to \{0, \ldots, m-1\}$$

  - $\mathcal{H}$ est universelle si $\forall l, k \in U$ avec $l \neq k$, le nombre de fonctions $h \in \mathcal{H}$ telles que $h(l) = h(k)$ vaut au plus $|\mathcal{H}|/m$

- Probabilité de collision : $1/m$

Conception d'une classe universelle de fct de hachage

exemple

- Conception d'une classe universelle avec les paramètres :
  - soit p un grand nombre premier tel que toute clé k est plus petite que p
  - soit $\mathtt{m}$ la taille de la table de hachage ($p > \mathtt{m}$)
  - soit $\mathbb{Z}_p = \{0, 1, \ldots, p-1\}$ et $\mathbb{Z}_p^* = \{1, \ldots, p-1\}$
  - soit $\mathrm{h}_{a,b}(\mathrm{k})$ une fonction de hachage pour tout $a \in \mathbb{Z}_p^*$ et tout $b \in \mathbb{Z}_p$:
$$\mathrm{h}_{a,b}(\mathrm{k}) = ((a\mathrm{k} + b) \bmod p) \bmod \mathtt{m}$$
- la classe $\mathcal{H}_{p,\mathtt{m}}$ est universelle :
$$\mathcal{H}_{p,\mathtt{m}} = \{\mathrm{h}_{a,b} : a \in \mathbb{Z}_p^* \text{ et } b \in \mathbb{Z}_p\}$$
  - il y a $p(p-1)$ fonctions de hachage différentes dans $\mathcal{H}_{p,\mathtt{m}}$
  - la probabilité de collision est $1/\mathtt{m}$

Sommaire

Adressage ouvert

principe

- Les éléments de l'ensemble dynamiques sont dans T
- T peut donc se remplir complètement, $m < |U|$
- La recherche d'une valeur se fait :
  - En suivant une chaîne de valeurs d'indices de T plutôt que suivant une liste chaînée
- L'insertion se fait en sondant la table de hachage T jusqu'à trouver une case vide

Adressage ouvert

séquence de sondages

- La fonction de hachage est étendue pour inclure le nombre de sondages, ainsi :
  - $h : U \times \{0, 1, \ldots, m-1\} \rightarrow \{0, 1, \ldots, m-1\}$
- La séquence de valeurs pour les sondages est :

$$\langle h(k,0), h(k,1), h(k,2), \ldots, k(k,m\text{-}1) \rangle$$

qui est une permutation de $\langle 0, 1, 2, \ldots, m\text{-}1 \rangle$ pour que toutes les cases puissent être visitées

insertion en adressage ouvert

Algorithme de `insertionHacahge(T,k)`

```
début
    i ← 0
    répéter
        j ← h(k,i)
        si T[j] = null alors
            retourner j
        sinon
            i ← i + 1
    jusqu'à i = m
    erreur
fin
```

- idem pour la recherche, mais attention à la suppression

Sondage linéaire

principe

- Soit $h' : U \to \{0, \ldots, m-1\}$ une fonction de hachage auxiliaire
- Soit la fonction de hachage linéaire h :
  $h(k, i) = (h'(k) + i) \bmod m \quad \text{avec} \quad i = 0, 1, \ldots, m-1$
  - première case sondée : `T[h'(k)]`
  - puis `T[h'(k)+1]`, ...

  Attention : problème de grappes
  - longue chaîne de cases occupées
  - si `i` cases pleines précèdent une case vide, la proba qu'elle soit la prochaine à être remplie est `(i+1)/m` au lieu de `1/m`

Sondage quadratique

principe

$$h(k, i) = (h'(k) + c_1 i + c_2 i^2) \bmod m$$

avec h' la fonction de hachage auxiliaire

$i = 0, 1, \ldots, m-1$, c_1 et $c_2 \neq 0$

- première case sondée : `T[k'(k)]`
- les accès suivants sont décalés suivant le numéro du sondage
  - mieux que le sondage linéaire
  - pour utiliser la table en entier, les valeurs de c_1, c_2 et m sont imposées
  - si $h(k_1, 0) = h(k_2, 0)$ alors $h(k_1, i) = h(k_2, i)\ \forall\ i$
  - grappe secondaire

Double hachage

principe

- Meilleure méthode que l'adressage ouvert

$$h(k,i) = (h_1(k) + i \times h_2(k)) \bmod m$$

avec h_1 et h_2 des fonctions de hachage auxiliaire

$i = 0, 1, \ldots, m-1$, c_1 et $c_2 \neq 0$

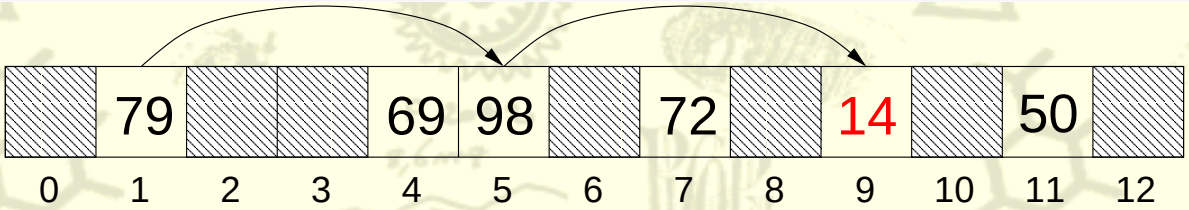
  - première case sondée : $T[h_1(k)]$
  - les accès suivants sont décalés de $h_2(k) \bmod m$
    - mieux que le sondage précédent
    - si $h(k_1,0) = h(k_2,0)$ alors $h(k_1,i) \neq h(k_2,i)\ \forall i$
    - le décalage varie en fonction de la clé

Exemple

fonction de double hachage

$$h(k, i) = (h_1(k) + i \times h_2(k)) \bmod m$$

- $m = 13$, $h_1(k) = k \bmod 13$ et $h_2(k) = 1 + (k \bmod 11)$



insertion de la clé 14

- $14 \equiv 1 \bmod 13$ et $14 \equiv 3 \bmod 11$
  - $h(14, 0) = h_1(14) = 1$ case occupée
  - $h(14, 1) = (h_1(14) + h_2(14)) \bmod 13 = 5$ case occupée
  - $h(14, 2) = (h_1(14) + 2 \times h_2(14)) \bmod 13 = 9$ case libre

➡ Insertion en case 9

Paramètres du double hachage

- $h_2(k)$ doit être premier avec m pour que T soit parcouru en totalité
  - exemple 1 : m une puissance de 2 et h_2 tjs impair
  - exemple 2 : m premier et h_2 telle que h soit positive inférieure à m
    - $h_1(k) = k \bmod m$
    - $h_2(k) = 1 + (k \bmod m')$ avec m' légèrement < à m
    - exemple : $k = 123456$ avec $m = 701$, $m' = 700$ on a $h_1(k) = 80$ et $h_2(k) = 257$
- On a m^2 séquences de sondages
- Schéma proche du hachage uniforme

Compléments

1. Thomas H. Cormen, Charles E. Leiserson, Ronald L. Rivest et Clifford Stein. *Introduction à l'algorithmique*, 2^{e} édition, Dunod, 2002.
2. Donald E. Knuth. *Sorting and Searching*, volume 3 de *The Art of Computer Programming*. Addison-Wesley, 1973.