7- Table de hachage

- Définitions
- Fonction de hachage
- Adressage chainé
- Adressage ouvert
- Réorganisation d'une table de hachage

7-0 : Rappel

Soient a et b deux entiers.
Soient q et r respectivement leur quotient et reste dans la division Euclidienne, on a :

$$a=b*q + r.$$

On dit que a est égal à r modulo b et on écrira

$$a=r[b].$$

7-1 Définitions

Définition 7.1 :
Soit **K** un ensemble de valeurs et **m** un entier naturel, une fonction de hachage h_m est une fonction définie de **K** dans **{0,…,m-1}**.

$$h_m : K \longrightarrow \{0,\ldots,m-1\}$$

- Les éléments de **K** sont appelées **clés**.
- Si **k** est une clé, **h(k)** est dite **valeur de hachage** de k.

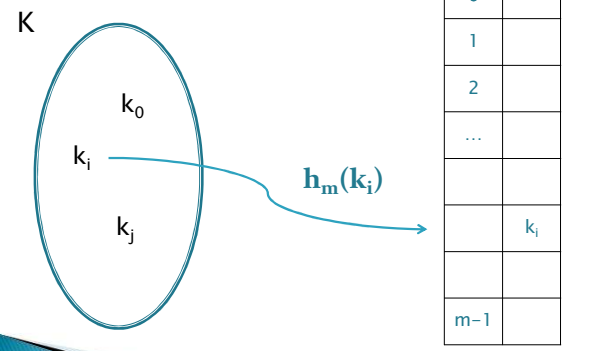
7-1 Définitions

Définition 7.2 :
Soit **m** un entier naturel et h_m une fonction de hachage, une **table de hachage** est un containeur tel que :

- le nombre d'éléments de la table (dimension ou taille) est fixe,
- l'accès aux éléments s'effectue indirectement par la valeur de hachage de la clé.

7-1 Définitions

- Exemple



7-1 Définitions

- La structure de données pour représenter une table de hachage est un tableau de dimension [0..m-1]. Pour une clé k de K, $T[h_m(k)]$ donne l'accès à l'élément de clé k.
- Plus qu'à la clé elle-même le plus souvent on souhaite accéder aux informations associées à la clé. La clé est donc stockée ainsi que l'adresse permettant d'accéder aux informations.
- On précisera ces moyens d'accès aux paragraphes concernant la gestion des collisions : **adressage chainé** , **adressage ouvert.**

7-1 Définitions

Remarques

- Les tables de hachage sont très utiles dans le cas où l'amplitude des clés est grande (structure de tableau non utilisable) et les éléments à gérer sont "assez figés" ce qui permet d'espérer un temps d'accès à l'information proche de O(1) (structure de liste trop pénalisante).
- Pour une séquence d'élément, il est clair qu'il peut exister deux clés k_1 et k_2 de K telles que $h_m(k_1)=h_m(k_2)$, on dit alors qu'il y a **collision** des clés k_1 et k_2.
- Soit p dans [0..m-1], il est possible qu'il n'existe pas de clé k dans K telle que $h_m(k)=p$.

7-1 Définitions

- On nommera le type abstrait *tableHash*.
- La valeur de hachage d'une clé est un curseur. Les primitives d'accès à une table de hachage sont :
- **accès**

```
fonction chercher(ref T:tableHash_de_cle, val v:cle):curseur;
```

- **modification**

```
fonction creerTablehachage(ref T: tableHash_de_cle,
                           ref h:fonction):vide;
fonction ajouter(ref T:tableHash de cle, val x:cle):booleen;
fonction supprimer(ref T:tableHash de cle, val :cle):vide;
fonction detruireTablehachage(ref T:tableHash de cle): vide;
```

Elles sont définies en fonction de h_m et du mode de gestion des collisions.

7-2 Fonction de hachage

Les fonctions de hachage doivent pouvoir s'appliquer sur des objets de n'importe quel type de base (entier, car, réel, etc.)

Les types numériques peuvent aisément être ramené à un entier. Pour les chaines de caractères, il est toujours possible de leur associer de manière bijective un entier.

Considérons la fonction asc qui a un caractère associe son code ASCII (codé sur 7 bits). Soit `c0...cp` une suite de p caractères alors la valeur entière associée bijectivement est :

$$\sum_{i=0}^{p} asc(c_i) * 128^i$$

Dans la suite on ne s'intéressera donc qu'à des clés entières.

7-2 Fonction de hachage

Famille de hachage universel

Définition 7.3 : Soit *H* un ensemble de fonction de hachage de *U* dans [0..m[. On dit que la famille **H est universelle** si pour toutes clés k_1, k_2 dans *U*, le nombre de fonctions *h* de H telles que $h(k_1)=h(k_2)$ est égal à $card(H)/m$.

⇒ Probabilité de collision : 1/m

7-2 Fonction de hachage

- **Méthode de division**

Soit m un entier premier pas trop proche d'une puissance de 2 :

$$\forall k \in \mathbb{N}, h_m(k) = k[m]$$

- **Méthode de multiplication**

Soit $m=2^p$ et A dans]0..1[,

$$\forall k \in \mathbb{N}, h_{m,A}(k) = \lfloor (m(kA - \lfloor kA \rfloor)) \rfloor$$

Cette méthode fonctionne mieux pour certaines valeurs de A que pour d'autres. D.E. Knuth a étudié le problème et suggère

$$A \sim \frac{\sqrt{5}-1}{2}$$

7-2 Fonction de hachage

Soit m un entier naturel. Soit p un nombre premier grand tel que :

$$\forall k \in K, k \in [0 \dots p-1]$$

On définit la famille de fonction : $H_{p,m}$

$$\forall k \in K, h_{a,b}(k) = ((ak+b)[p])[m]$$

Théorème 7.1 : La famille de fonction $H_{p,m}$ *est universelle*

De ce fait, on peut montrer que l'exécution de n primitives a une complexité moyenne en O(n).

7-3 Adressage chainé

Définition 7.4 : L'adressage d'une table de hachage est dit chainé si l'élément i du tableau donne accès à une structure de données permettant de stocker les clés k telles que $h_m(k)=i$.

Les valeurs sont donc stockées à l'extérieur de la table qui est un tableau de pointeurs.
Par suite, si il n'existe pas de clé k telle que $h_m(k)=i$ alors T[i]=NIL.

L'espace de stockage des clés peut être une liste doublement chainée.
On a dans ce cas

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

7-3 Adressage chainé

Primitives de modification :

```
fonction creerTableHach(ref T: tableHash_de_cle,
                        ref h:fonction):vide;
  var i:entier;
  debut
    pour i allant de 0 à m-1 faire
      creerListe(T.table[i])
    finpour
    T.h=h;
  fin

fonction ajouter(ref T:tableHash_de_cle,val e:entier):booleen;
       debut
              insererEnTete(T.table[T.h(e)],e);
              retourner(vrai):
       fin
```

7-3 Adressage chainé

Primitives d'accès :

```
fonction chercher(ref T:tableHash_de_cle,
                  val e:entier):curseur;
  debut
    retourner(chercherListe(T.table[T.h(e)],e))
  fin
```

7-3 Adressage chainé

Primitives de modification :

```
fonction supprimer(ref T:tableHash_de_cle;val e:entier):vide;
  debut
    supprimer(T.table[T.h(e)],e))
  fin
```

Théorème 7.2 : Dans une table de hachage à adressage chainé, une recherche fructueuse ou infructueuse prend en moyenne $O(1+n/m)$ si chaque élément à les mêmes chances d'être haché vers l'une quelconque des cases indépendamment des autres éléments (hachage uniforme).

7-4 Adressage ouvert

Définition 7.5 : L'adressage d'une table de hachage est dit ouvert si les éléments du tableau sont les clés elles mêmes.

Les éléments de la table sont donc initialisés à NULL.
La gestion de la collision se fait en trouvant une place libre dans la table. Pour cela on utilise une seconde fonction `s(x,i)` à valeur dans `[0..m]` que l'on compose avec $h_m(k)$.

7-4 Adressage ouvert

Quelques méthodes de sondage :

- Sondage linéaire :
  $s(k,i)=(h(k)+i)[m]$
- Sondage quadratique :
  $s(k,i)=(h(k)+c_1\ i+c_2\ i^2)[m]$
- Double hachage :
  $s(k,i)=(h(k)+i\ h'(k))[m]$

où h' est une seconde fonction de hachcode telle que
$\forall k \in K, h'(k) est\ premier\ avec\ m$

7-4 Adressage ouvert

Cette seconde fonction dite de **sondage** doit vérifier les propriétés suivantes :

- $s(h_m(k),0)=h_m(k)$
- $s(h_m(k),i))_{i=0...m-1}$ est une permutation de la séquence (i) $_{i=0...m-1}$

La méthode d'insertion consiste, en cas de collision pour une clé k, à sonder les places disponibles $s(h_m(k),i)$ à partir de i=0 jusqu'à
m-1.

Si au bout de m sondages, aucune place disponible n'a été détectée, la table est dite **saturée**.

7-4 Adressage ouvert

L'espace de stockage des clés est alors un tableau.
On a dans ce cas :

```
tableHash_de_cle=structure
     table:tableau[0..m-1] de  cle;
     h : fonction(val v : cle) : entier
     s : fonction(val v : cle; val i : entier) : entier
 finstructure
```

7-4 Adressage ouvert

Primitives d'accès :

```
fonction chercher(ref T:tableHash_de_cle,
                  val e:entier):curseur;
  var i:entier;
  debut
     i=0;
     tant que T.table[T.s(T.h(e),i)]!=e et i<m faire
       i=i+1
     fintantque
     si i==m alors
       retourner(NULL)
     sinon
       retourner(i)
     finsi
  fin
```

7-4 Adressage ouvert

Primitives de modification

```
fonction ajouter(ref T:tableHash_de_cle,
      val e :entier):booleen;
  var i:entier;
  debut
     i=0;
     tant que T.table[T.s(T.h(e),i)]!=NULL et i<m faire
       i=i+1
     fintantque
     si i==m alors
       retourner(faux)
     sinon
       T.table[T.s(T.h(e),i)]=e;
       retourner(vrai)
     finsi
  fin
```

7-4 Adressage ouvert

Primitives de modification

```
fonction creerTablehachage(ref T: tableHash_de_cle,
      ref h : fonction(var x:entier):entier),
      ref s : fonction(var x:entier):entier):vide;
  var i:entier;
  debut
    pour i allant de 0 à m-1 faire
      T.table[i]=NULL;
    finpour
    T.h=h;
    T.s=s;
  fin
```

7-4 Adressage ouvert

Primitives de modification

```
fonction supprimer(ref T:tableHash_de_cle,
                   val e:entier):vide;
  var p:curseur;
  debut
    p=chercher(T,e);
    T.table[p]=NULL
  fin
```

7-4 Adressage ouvert

Théorème 7.3 : Etant donné une table de hachage de facteur de remplissage r, une recherche infructueuse ou l'insertion d'une clé se fait en moyenne en 1/(1−r), si chaque élément a les mêmes chances d'être haché vers l'une quelconque des cases indépendamment des autres éléments (hachage uniforme).

Théorème 7.4 : Etant donné une table de hachage de facteur de remplissage r, une recherche fructueuse se fait en moyenne en (1/r)*ln(1/(1−r)), si chaque élément a les mêmes chances d'être haché vers l'une quelconque des cases indépendamment des autres éléments (hachage uniforme).

7-5 Réorganisation d'une table de hachage

Le **taux d'occupation** de la table doit rester relativement **faible** pour minimiser le risque de collision sinon le nombre de collision augmente et la performance se dégrade.

Ceci veut dire qu'on ajoute à chaque type abstrait un champ "taux de remplissage" égal au rapport du nombre d'éléments divisé par la taille de la table.

Dès que ce taux dépasse un seuil fixé, la fonction "ajouter" effectue automatiquement l'appel à la fonction de redimensionnement de la table.