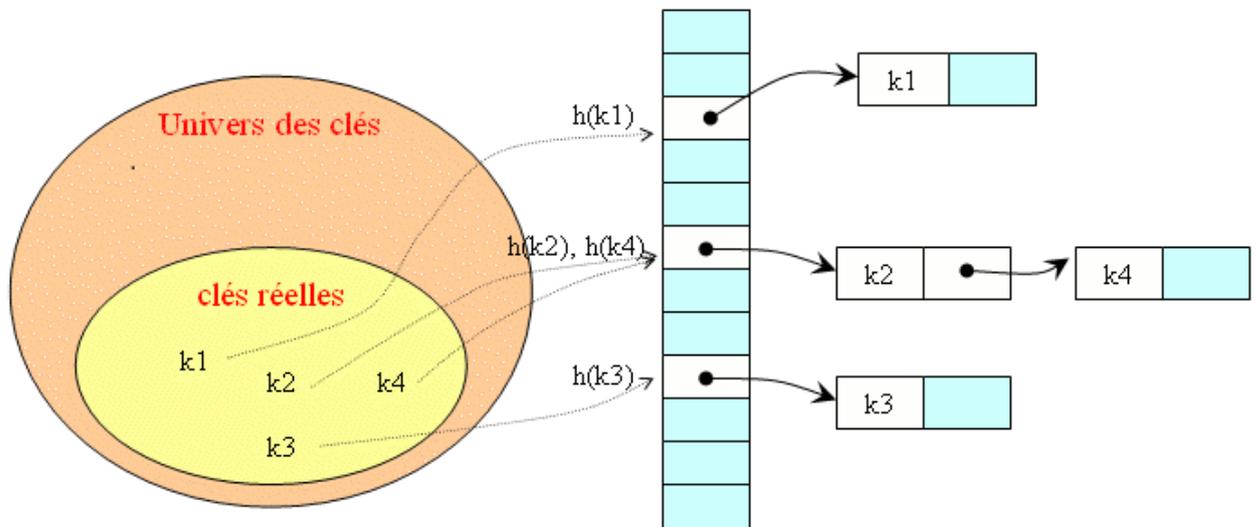


## 3 Le chaînage

### 3.1 Principe

#### Résolution des collisions par chaînage

Avec cette technique, on place dans une liste chaînée tous les éléments ayant la **même valeur de hachage** :



Pour rechercher un élément, il suffit à présent de parcourir la liste chaînée dont l'accès se trouve dans la case d'indice  $h(x)$ . L'avantage est l'aisance de l'implémentation, et de plus, il n'y a pas de limitation (théorique) au nombre de clés. Toutefois, il faut veiller à prendre un tableau assez grand pour que la taille des listes reste réduite, le but étant de minimiser les parcours.

#### Exemple

L'introduction des clés 12, 17, 29, 33, 42 et 39 dans la table donnerait la configuration suivante :

1	2	3	4	5	6	7	8	9	10
Nil	Nil			Nil	Nil	Nil		Nil	
		↓	↓				↓		↓
		42	33				17		39
		↓							↓
		12							29

(Il est évident que les ajouts se feront toujours en tête de liste).

## 3.2 Opérations de dictionnaire

Les opérations de dictionnaire sont implémentées facilement :

- `insérer(x)` : insertion d'un élément  $x$  en tête de la liste chaînée  $h(x.valeur)$
- `chercher(k)` : recherche d'un élément de valeur  $k$  dans la liste chaînée  $h(k)$
- `supprimer(x)` : suppression d'un élément  $x$  dans la liste chaînée  $h(x.valeur)$

Si l'opération de hachage est  $O(1)$  alors l'insertion est en  $O(1)$  mais la complexité de la recherche et de suppression sont indéfinies. On introduit la notion de facteur de remplissage.



### Facteur de remplissage

Étant donné une table de hachage  $T$  avec  $m$  cases et contenant  $n$  éléments, le **facteur de remplissage** est défini par :

$$\alpha = \frac{n}{m}$$



### Remarque

Le comportement dans le pire cas est en  $\Theta(n)$  si tous les objets sont dans la même case. On suppose qu'un objet a une chance équitable de se retrouver dans une des cases : le hachage est *uniforme simple*.

## 3.3 Complexité de la recherche



### Théorème

Dans une table de hachage avec résolution des collisions par chaînage, une recherche qui échoue prend en moyenne un temps  $\Theta(1 + \alpha)$  avec une fonction de hachage uniforme simple.

### Preuve

Avec un hachage uniforme simple, toute valeur a la même probabilité d'être hachée vers une case  $m$ . Le temps moyen d'une recherche infructueuse est donc le temps moyen de parcours d'une des  $m$  listes chaînées. La longueur moyenne de ces listes étant le facteur de remplissage, il y a donc en moyenne  $\alpha$  éléments à examiner. Rappel : le calcul de la fonction de hachage est en  $O(1)$ . ■



### Théorème

Dans une table de hachage avec résolution des collisions par chaînage, une recherche réussie prend en moyenne un temps  $\Theta(1 + \alpha)$  avec une fonction de hachage uniforme simple.

**Preuve**

Pour simplifier la preuve, on suppose que l'on insère toujours en fin de la liste chaînée. Si on cherche le  $i$ -ème élément inséré, il nous faut parcourir la chaîne sur une longueur égale à celle qu'elle avait sans cet élément puis effectuer une comparaison. Quand l'élément  $i$  a été ajouté, la chaîne avait une longueur  $\frac{i-1}{m}$ . Rechercher le  $i$ -ème élément demande donc  $1 + \frac{i-1}{m}$  opérations. En faisant la moyenne :

$$\begin{aligned} \frac{1}{n} \sum_{i=1}^n \left(1 + \frac{i-1}{m}\right) &= 1 + \frac{1}{nm} \sum_{i=1}^n (i-1) \\ &= 1 + \frac{1}{nm} \left(\frac{(n-1)m}{2}\right) \\ &= 1 + \frac{\alpha}{2} - \frac{1}{2m} \end{aligned}$$

Si l'on considère que le nombre de cases  $m$  est proportionnel au nombre d'éléments  $n$ , on a  $n = O(m)$  et  $\alpha = n/m = O(m)/m = O(1)$ . ■

**Conclusion**

Il est possible d'avoir une recherche en  $O(1)$  en moyenne dans une table utilisant des listes chaînées mais il faut maintenir la relation  $n = O(m)$ , autrement dit augmenter/diminuer la taille de la table tout le temps. En pratique, on ne le fait pas, mais on est très poche de  $O(1)$  quand même.

### 3.4 Complexité de la suppression

La complexité de la suppression dépend de l'information dont on dispose pour retirer l'objet (valeur ou référence vers l'objet) et de la façon dont est gérée la liste chaînée (simple ou double).

En considérant  $\alpha$  le nombre d'éléments dans la liste, et si on ne dispose que de la valeur, il faut trouver l'objet en  $O(\alpha)$  puis le supprimer. Rappel (si on a une référence sur un élément) :

- Supprimer dans une liste simplement chaînée :  $O(\alpha)$
- Supprimer dans une liste doublement chaînée :  $O(1)$

### 3.5 Conclusion : Complexité moyenne

On a donc les complexités suivantes :

- Insertion :  $\Theta(1)$
- Recherche :  $\Theta(1 + \alpha)$
- Suppression :  $\Theta(1 + \alpha)$

Si le nombre de cases est proportionnel au nombre d'éléments, alors toutes les opérations peuvent être supportées en  $O(1)$ .

### 3.6 Exemple de fonction de hachage

Idéalement une fonction de hachage devrait vérifier l'hypothèse de hachage uniforme. En pratique, on essaie de s'en approcher au maximum.



#### Méthode de la division

Elle fait correspondre une valeur  $k$  avec une des  $m$  alvéoles, en prenant le reste de la division de  $k$  par  $m$  :

$$h(k) = k \bmod m$$



#### Remarque

Cette méthode est très rapide mais certaines valeurs de  $m$  sont à éviter.

#### Exemple

Prenons l'ensemble des étudiants d'une université, avec comme clé le numéro d'étudiant de 5 chiffres. La fonction  $h(x) = x \text{ div } 1000$  ne serait pas un bon choix car elle donnerait un nombre réduit de valeurs (comprises actuellement entre 35 et 40), à partir d'un ensemble de plusieurs centaines d'étudiants. La fonction  $h(x) = x \bmod 100$  est déjà bien meilleure, elle donnerait des valeurs entre 0 et 99. Il y aura bien sûr des collisions, vu que les étudiants de numéros 37156 et 38956 seraient « hachés » de la même façon.

#### Exemple

Pour les codes postaux des villes de Belgique, on pourrait prendre la fonction  $h(x) = x \bmod 10$ . Elle donnerait un ensemble de valeurs entre 100 et 999 où les collisions seraient peu nombreuses (vu que la plupart des codes se terminent par 0).

## 4 Adressage ouvert

### 4.1 Principe

#### Résolution des collisions par l'adressage ouvert

L'idée est la suivante : lorsqu'on veut utiliser un emplacement de la table et que celui-ci est occupé, on va voir ailleurs. Dans la technique la plus simple, on va simplement continuer le parcours du tableau, à partir de la position de départ, à la recherche d'un « trou ». Pour la recherche, on procède de même : la fonction de hachage nous indique où commencer à chercher et on poursuit jusqu'à trouver l'élément ou un « trou ».

#### Exemple

Reprenons l'exemple ci-avant : la clé 42 devrait occuper la case d'indice 3. Comme celle-ci est occupée, on parcourt le tableau jusqu'à la prochaine case libre, celle d'indice 5 et on y place l'élément :

1	2	3	4	5	6	7	8	9	10
		12	33	42			17		29

Ce parcours de recherche est « circulaire » : arrivé à la dernière case, on revient au début. Ainsi, la clé 39 qui ne peut pas être mise à sa place « normale » (occupée par 29) sera placée en première position :

1	2	3	4	5	6	7	8	9	10
39		12	33	42			17		29

L'exemple peut laisser perplexe vu la petite taille du tableau ; si le tableau a une taille suffisante, on peut imaginer que la clé cherchée ne sera jamais très loin de la position donnée par  $h(x)$  de telle façon que la longueur du parcours requis peut être considéré comme négligeable.

La suppression est plus délicate à mettre en oeuvre : il ne suffit pas de supprimer simplement la clé dans la case donnée par la fonction de hachage, car s'il y a eu une collision, on ne retrouverait plus l'autre valeur de clé. Il faut donc reboucher le trou avec la dernière clé donnant la même valeur de hachage. Ainsi, dans l'exemple, si on supprime 12, il faut déménager 42 dans la case d'indice 3 :

1	2	3	4	5	6	7	8	9	10
39		42	33				17		29

#### Conclusion

Notons que la table a aussi une capacité maximale, on ne pourra pas y mettre plus de clés que sa taille. Cette technique est donc assez délicate mais intéressante lorsque les allocations dynamiques sont impossibles ou coûteuses. Si elles sont possibles, on utilisera plutôt la technique du chaînage.

## 4.2 Opérations de dictionnaire

### Opération d'insertion

Pour effectuer une insertion, on examine (sonde) successivement les cases de la table jusqu'à en trouver une vide. La séquence de positions à tester dépend de la clé à insérer.

On étend la fonction de hachage pour y inclure le nombre de sondages déjà effectués :

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

Pour examiner toutes les cases, il faut que la séquence de sondage soit une permutation de l'ensemble des valeurs de hachage. L'opération d'insertion s'écrit :



### Algorithme inserer

Fonction `inserer` ( DR T : OpenHash ; x : Elément ) : Entier

Variable k : Entier

Début

```
| k <- clé ( x )
| i <- 0
| Répéter
|   | j <- h ( k , i )
|   | Si T [ j ] = Nil Alors
|   |   | T [ j ] <- x
|   |   | Retourner j
|   | Sinon
|   |   | i <- i + 1
|   | FinSi
| Jusqu'à i = m
| Erreur "Plus de place"
```

Fin

### Opération de recherche

L'opération de recherche s'écrit :



### Algorithme rechercher

Fonction `recherche` ( T : OpenHash ; x : Elément ) : Itérateur

Variable k : Entier

Début

```
| k <- clé ( x )
| i <- 0
| Répéter
|   | j <- h ( k , i )
|   | Si T [ j ] = Nil Alors
|   |   | Retourner j
|   | Sinon
|   |   | i <- i + 1
|   | FinSi
| Jusqu'à ( T [ j ] = Nil Ou i = m )
| Retourner Nil
```

Fin

### 4.3 Complexité de la recherche



#### Théorème

Dans une table de hachage en adressage ouvert avec un facteur de remplissage  $\alpha = n/m < 1$ , le nombre de sondages lors d'une recherche infructueuse vaut au plus  $1/(1-\alpha)$  avec une fonction de hachage uniforme.

#### Preuve

Étudions le pire cas d'une recherche infructueuse.

- Chaque sondage sauf le dernier accède à une case ne contenant pas la valeur recherchée.
- Le dernier accède à une case vide.
- Pourquoi la dernière case ne peut pas être pleine ?

On définit

$$p_i = \Pr\{i \text{ sondages exactement accèdent à des alvéoles occupées}\}, i = 0, 1, 2, \dots$$

Pour  $i > n$ , on a  $p_i = 0$  car on ne peut pas trouver plus de  $n$  alvéoles occupées.

Le nombre de sondages attendues est donc :

$$1 + \sum_{i=0}^{\infty} ip_i$$

Pour calculer cette formule, on utilise un résultat de probabilité. Quand une variable aléatoire  $X$  prend une valeur dans l'ensemble des naturels, on a :

$$\begin{aligned} E[X] &= \sum_{i=0}^{\infty} i \Pr(X = i) \\ &= \sum_{i=0}^{\infty} i(\Pr(X \geq i) - \Pr(X \geq i + 1)) \\ &= \sum_{i=0}^{\infty} \Pr(X \geq i) \end{aligned}$$

en remarquant que chaque terme  $\Pr(X \geq i)$  est ajouté  $i$  fois et soustrait  $i - 1$  fois, sauf pour le premier terme.

On introduit :

$$q_i = \Pr\{i \text{ sondages au moins accèdent à des alvéoles occupées}\}, i = 0, 1, 2, \dots$$

On a donc (cf. résultat précédent)

$$\sum_{i=0}^{\infty} ip_i = \sum_{i=1}^{\infty} q_i$$

Que vaut  $q_i$  ?

- $q_1$  est la probabilité pour que le premier sondage bute sur une case occupée, donc :

$$q_1 = \frac{n}{m}$$

- Avec un hachage uniforme simple, le deuxième sondage se fera uniquement sur une nouvelle case, donc parmi  $m - 1$  cases. On effectue le deuxième sondage que si le premier bute sur une case occupée, d'où :

$$q_2 = \binom{n}{m} \binom{n-1}{m-1}$$

- On en déduit :

$$\begin{aligned} q_i &= \binom{n}{m} \binom{n-1}{m-1} \cdots \binom{n-i+1}{m-i+1} \\ &= \leq \left(\frac{n}{m}\right)^i \\ &= \leq \alpha^i \end{aligned}$$

Finalement en utilisant l'hypothèse  $\alpha < 1$ , on a

$$\begin{aligned} 1 + \sum_{i=0}^{\infty} ip_i &= 1 + \sum_{i=1}^{\infty} q_i \\ &\leq 1 + \alpha + \alpha^2 + \dots \\ &\leq \frac{1}{1 - \alpha} \end{aligned}$$

Ce qui conclut la preuve. ■

#### 4.4 Exemple de fonction de hachage

Cette fois, la fonction de hachage ne doit pas seulement retourner une valeur, mais elle doit donner une série de valeurs (toutes les alvéoles).



##### Méthode de sondage linéaire

Elle est classiquement utilisée :

$$h(k, i) = (h'(k) + i) \pmod{m}$$

avec  $h'(k)$  une fonction de hachage ordinaire. Bien sûr, cette méthode a ses défauts.