


CHAPITRE 9

HACHAGE STATIQUE


Chapitre 9 - Hachage Statique 1



Hachage Statique

1. Introduction
2. Principe du hachage
3. Terminologie
4. Fonctions de Hachage
5. Méthodes de résolution de collisions
 - essai linéaire – double hachage
 - chainage séparée – chainage interne
6. Comparaison entre les différentes méthodes
7. Estimation des débordements
8. Conclusion

Chapitre 9 - Hachage Statique 2



1. Introduction

Problématique


Supposons que nous avons n enregistrements (données) stockés dans un tableau. On désire:

- rechercher une donnée x
- insérer une donnée x

Solutions

- * Si tableau **ordonné** Alors
 - Recherche: Recherche Dichotomique → $O(\log(n))$ → **Rapide**
 - Insertion: Décalage des éléments du tableau → **Lent**
- * Si tableau **non ordonné** Alors
 - Recherche: Recherche Séquentielle → $O(n)$ → **Lent**
 - Insertion: Insertion fin du tableau → **Rapide**

Chapitre 9 - Hachage Statique 3



1. Introduction

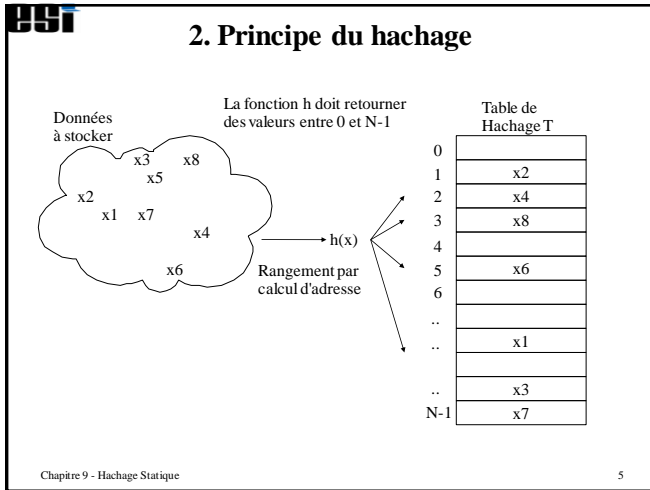
Autre solution

Tableau est une table de hachage dont la complexité de ces opérations (recherche et insertion), en moyenne, peut se faire en un temps constant

Hachage = Hashing (en anglais)

= Technique de rangement dispersé

Chapitre 9 - Hachage Statique 4



- ## 3. Terminologie
- Stocker des données (x) dans une table (T) de taille N , en utilisant une fonction (h) pour la localisation rapide
→ **calcul d'adresse**
 - La fonction (h) calcule l'emplacement de (x) et retourne la case d'indice
→ **adresse primaire**
 - Si la case est déjà occupée (**collision**), on insère (x) à un autre emplacement
→ **adresse secondaire** → **Débordement**
 - L'adresse secondaire est déterminé par un algorithme donné
→ **méthode de résolution de collisions**
 - Si des données ont la même image par la fonction de hachage
→ **synonyme**
- Chapitre 9 - Hachage Statique 6

En résumé, pour utiliser une table de rangement dispersé (hachage), on doit donc définir:

- une fonction de hachage
- une méthode de résolution des collisions

Chapitre 9 - Hachage Statique 7

- ## 4. Fonctions de hachage
- Il s'agit de trouver une fonction h tels que:
 $0 \leq h(x) < N$
qui réduit au maximum le nombre de collisions
 - L'idéal, c'est d'avoir une fonction de hachage bijective.
 - Le pire des cas, c'est lorsque toute donnée est hachée en une même adresse.
 - Une solution acceptable est une solution où certaines données partagent la même adresse (f est surjective).
 - Quelques fonctions de hachage usuelles
 - La fonction de division
 - La fonction du « middle square »
 - La fonction du « transformation radix »
- Chapitre 9 - Hachage Statique 8

PSI

• **RAPPEL:**

SURJECTION

Une application f de X dans Y est dite surjective si tout élément de l'ensemble d'arrivée Y a au moins un antécédent par f , c'est-à-dire si :

$$\forall y \in Y, \exists x \in X, f(x) = y$$

INJECTION

Une application f de X dans Y est dite injective si :

$$\forall (x, y) \in X^2, (x \neq y \Rightarrow f(x) \neq f(y))$$

BIJECTION

Une application f de E dans F est dite bijective si et seulement si tout élément de l'ensemble d'arrivée F a exactement un antécédent par f dans l'ensemble de départ E c'est-à-dire si :

$$\forall y \in F, \exists! x \in E / f(x) = y$$

Chapitre 9 - Hachage Statique 9

PSI

4. Fonctions de hachage

1) La fonction de division

$h(x) = x \text{ MOD } N$

retourne le reste de la division par N (la taille de la table)

- C'est une fonction facile et rapide à calculer mais sa qualité dépend de la valeur de N .
- Il est préférable que N soit **premier** et ne doit pas être une puissance de 2

Chapitre 9 - Hachage Statique 10

PSI

4. Fonctions de hachage

1) La fonction de division

Exemple: (voir figure au tableau)

Soit $N = 10$ Alors:

$h(5) = 5 \text{ MOD } 10 = 5$

$h(55) = 55 \text{ MOD } 10 = 5 \rightarrow \text{collision}$

$h(23) = 23 \text{ MOD } 10 = 3$

$h(453) = 453 \text{ MOD } 10 = 3 \rightarrow \text{collision}$

Soit $N = 11$ Alors:

$h(5) = 5 \text{ MOD } 11 = 5$

$h(55) = 55 \text{ MOD } 11 = 0$

$h(23) = 23 \text{ MOD } 11 = 1$

$h(453) = 453 \text{ MOD } 11 = 2$

Pas de collisions dans ce cas car $N=11$ est un nombre **premier**

→ **minimiser les collisions**

Chapitre 9 - Hachage Statique 11

PSI

4. Fonctions de hachage

2) La fonction du milieu du carré « middle square »

On élève la donnée x au carré x^2 et on prend les chiffres du milieu

- Cette méthode donne de bons résultats si le nombre au carré n'a pas de zéros.

Exemple:

Soit $N = 100$

$h(500) = 0$ car $(500)^2 = 250000$

$h(12) = 14$

ou

$h(12) = 44$ car $(12)^2 = 144$

Chapitre 9 - Hachage Statique 12



4. Fonctions de hachage

3) La fonction dite « transformation radix »

On convertit la donnée x dans une base de numération et on prend le reste de la division.

Exemple:

Soit: $x = (453)_{10} \rightarrow$ base 10

$x = (382)_{11} \rightarrow$ base 11

$h(x) = (x)_{11} \text{ MOD } N$



4. Fonctions de hachage

En conclusion

Il n'y a pas de fonction de hachage universelle.

Cependant, une bonne fonction doit être:

- rapide à calculer
- répartit uniformément les éléments

Elle dépend donc:

- de la machine
- des éléments

Mais aucune fonction n'évite les collisions, qu'il va falloir traiter.



5. Méthodes de résolution de collisions

- Lors de l'insertion de x , si l'adresse primaire $h(x)$ est déjà utilisée par une autre donnée, la méthode de résolution de collision permet de trouver un autre emplacement (libre) pour x .
- Pour résoudre les collisions, deux stratégies se présentent:
 - Les méthodes directes ou le hachage par calcul de l'emplacement:
 - 1) Essai linéaire
 - 2) Double hachage
 - Les méthodes indirectes ou le hachage par chaînage:
 - 3) Chaînage séparée
 - 4) Chaînage interne



5. Méthodes de résolution de collisions

1) Essai Linéaire

Principe:

- S'il se produit une collision sur la case $h(x)$, on essaie les cases qui la précèdent : $h(x)-1$, $h(x)-2$, $h(x)-3, \dots$, 0, $N-1$, $N-2$, ..., jusqu'à trouver une case vide.
 - La rencontre d'une case vide indique que la donnée n'existe pas.
- ➔ Il faudra sacrifier une case vide dans la table de hachage pour que la séquence de test soit finie.

5. Méthodes de résolution de collisions

1) Essai Linéaire

Exemple:

L'insertion des données suivantes, donne la table ci-après.
Le bit 'vide' à 1, indique une case est vide.

$h(a) = 5$

$h(b) = 1$

$h(c) = 3$

$h(d) = 3 \rightarrow$ collision

Calcul de $h(d) - 1 = 2 \rightarrow$ case vide

	Donnée	vide
0		1
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8		1
9		1
10		1

Chapitre 9 - Hachage Statique

17

5. Méthodes de résolution de collisions

1) Essai Linéaire

$h(e) = 0$

$h(f) = 2 \rightarrow$ collision

Calcul de $h(f) - 1 = 1 \rightarrow$ case pleine

Calcul de $h(f) - 2 = 0 \rightarrow$ case pleine

Calcul de $h(f) - 3 = -1$

// si $(h(x) - i < 0)$ alors calcul de $(h(f) - i) + N$

calcul de $h(f) - 3 + 11 = 10 \rightarrow$ case vide

$h(g) = 8$

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

Chapitre 9 - Hachage Statique

18

5. Méthodes de résolution de collisions

1) Essai Linéaire

$h(a) = 5 \rightarrow$ adresse primaire

$h(b) = 1 \rightarrow$ adresse primaire

$h(c) = 3 \rightarrow$ adresse primaire

$h(d) = 3 \rightarrow$ adresse secondaire (collision)

$h(e) = 0 \rightarrow$ adresse primaire

$h(f) = 2 \rightarrow$ adresse secondaire (collision)

$h(g) = 8 \rightarrow$ adresse primaire

• La recherche de x (tel que $h(x) = 2$) s'arrête avec un échec dans la case vide d'adresse 9

\Rightarrow la séquence de test est : 2,1,0,10,9

• Si on devait insérer x, la donnée serait affectée à la case 9 (si c'est pas la dernière case vide).

La table est remplie quand le nombre d'éléments insérés égale à $N-1 \rightarrow$ sacrifice d'une case vide

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

Chapitre 9 - Hachage Statique

19

5. Méthodes de résolution de collisions

1) Essai Linéaire: Algorithme d'insertion d'un élément

L1. [Hacher]

$i := h(K) \{ 0 \leq i < M \}$

L2. [Comparer]

SI Donnée(K) = K, l'algorithme se termine avec succès.

Autrement SI T(i) est vide aller à L4.

L3. [Avancer au prochain]

$i := i - 1$

SI $i < 0$ Alors $i := i + N$

ALLER A L2.

L4. [Insérer] {la recherche est sans succès}

SI $M = N - 1$ Alors

l'algorithme se termine avec débordement

SINON

$M := M + 1$ {Nombre d'éléments insérés}

Marquer T(i) occupé

Donnée(i) := K

Chapitre 9 - Hachage Statique

20

5. Méthodes de résolution de collisions

1) Essai Linéaire

- La suppression d'un élément x, **génère une case vide**

→ **suppression physique**

- Cette nouvelle case vide risque de rendre d'autres **données inaccessibles**.

Dans l'exemple précédent, si on supprime b en vidant la case 1, on perd du même coup la donnée f (car elle n'est plus accessible)

→ **faire des tests avant de vider une case**

	Donnée	vide
0	e	0
1	b	0
2	d	0
3	c	0
4		1
5	a	0
6		1
7		1
8	g	0
9		1
10	f	0

5. Méthodes de résolution de collisions

1) Essai Linéaire

- Le **principe de la suppression** d'une donnée x est donc :

- Rechercher l'adresse j de x
- Tester toutes les cases ($i=j-1, j-2, \dots$) au dessus de j (circulairement) jusqu'à trouver une case vide et **vérifier** que les données qu'elles contiennent ne vont pas être perdues quand la case j sera vidée
- Si c'est le cas, on vide la case j et on s'arrête
- Sinon, dès qu'on trouve une donnée «**qui pose problème**» on la **déplace** dans la case j et on tente de vider son emplacement (en testant les cases au dessus qui n'ont pas encore été testées). C'est le même principe qu'on vient d'appliquer pour la case j.

5. Méthodes de résolution de collisions

1) Essai Linéaire: Algorithme de suppression d'un élément

Soit i l'adresse de l'élément à supprimer:

- Rendre T(i) vide
Poser j=i

- $i=i-1$
si $i < 0$ alors $i = i + N$

- si T(i) est vide alors
Algo se termine
sinon

soit $r=h(T(i))$

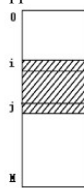
a) $i < j$

si $r < i$ ou $r \geq j$ alors déplacer l'élément ($T(j) = T(i)$)

b) $i > j$

si $j \leq r < i$ alors déplacer l'élément ($T(j) = T(i)$)

- Recommencer à partir de 2.



a)



b)

5. Méthodes de résolution de collisions

2) Double Hachage

- Cette méthode est presque analogue à la précédente mais au lieu que la séquence soit linéaire, elle est construite par une autre fonction de hachage soit h' .
- Soient $h(x)$ la fonction utilisée pour le calcul de l'adresse primaire et $h'(x)$ la seconde fonction de hachage qui calcule le pas de la séquence: $h(x), h(x)-h'(x), h(x)-2h'(x), h(x)-3h'(x), \dots$
- Pour que la séquence soit circulaire, les soustractions se font modulo N (la taille de la table) c'est-à-dire quand on calcule le nouvel indice $i := i - h'(x)$, on rajoute juste après le test:

SI ($i < 0$) Alors $i := i + N$

5. Méthodes de résolution de collisions

2) Double Hachage

- Pour que la couverture soit totale (passer par toutes les cases), il faut choisir la taille N de la table un **nombre premier**
- Pour simplifier les algorithmes, on sacrifie une case de la table càd il reste toujours au moins une case vide (critère d'arrêt)

5. Méthodes de résolution de collisions

2) Double Hachage

Exemple:

- Soit T une table de 6 éléments.
- Le bit 'vide' à 1, indique une case vide.
- L'insertion des données suivantes d'après la première fonction de hachage h:

$$\begin{array}{lll} h(a) = 3 & h(b) = 2 & h(c) = 3 \\ h(d) = 2 & h(e) = 1 & \end{array}$$

et en plus avec h' la deuxième fonction de hachage:

$$h'(c) = 3 \quad h'(d) = 1 \quad h'(e) = 3$$

5. Méthodes de résolution de collisions

2) Double Hachage

$h(a) = 3 \rightarrow$ inséré à la case 3

$h(b) = 2 \rightarrow$ inséré à la case 2

$h(c) = 3 \rightarrow$ **collision** à la case 3
calcul de $h(c) - h'(c) = 0$ (avec $h'(c) = 3$)
inséré à la case 0

$h(d) = 2 \rightarrow$ **collision** à la case 2
calcul de $h(d) - h'(d) = 1$ (avec $h'(d) = 1$)
inséré à la case 1

$h(e) = 1 \rightarrow$ **collision** à la case 1
calcul de $h(e) - h'(e) = 1 - 3 = -2$ (avec $h'(e) = 3$)
soit $i = h(e) - h'(e)$
si $i < 0$ alors $i = i + N$ donc $i = 4$ (avec $N = 6$)
inséré à la case 4

0	c
1	d
2	b
3	a
4	e
5	

Table T

5. Méthodes de résolution de collisions

2) Double Hachage: Algorithme d'insertion d'un élément

- D1. [premier hachage]
 $i := h(K)$
- D2. [premier test]
SI T(i) vide Alors ALLERA D6.
SI Donnée(i) = K Alors l'algorithme se termine avec succès.
- D3. [second hachage]
 $c := h'(K)$
- D4. [Avancer au prochain]
 $i := i - c$
SI $i < 0$ Alors $i := i + N$
- D5. [Comparer]
SI T(i) est vide Alors ALLERA D6.
SI Donnée(i) = K Alors l'algorithme se termine avec succès.
SINON ALLERA D4
- D6. [Insérer]
SI $N = M - 1$ Alors "débordement"
SINON $M := M + 1$ {Nombre d'éléments insérés}
Rendre T(i) occupé
Donnée(i) := K

5. Méthodes de résolution de collisions

2) Double Hachage

Le principe de la suppression est analogue à celui de l'essai linéaire.

Un moyen simple consiste à faire une suppression logique, c'est à dire le positionnement d'un bit qu'on rajoute au niveau des entrées de la table.

Chaque case renferme 2 bits :

vide indiquant une case vide

eff indiquant un effacement logique (la case n'est pas vide)
ex: b est supprimée logiquement

	donnée	eff	vide
0			
1	c	0	0
2			
3	b	1	0
4			
5	a	0	0
6			
7	d	0	0
8			
N-1			

Pour récupérer l'espace perdu à cause des effacements logiques, on effectue des **réorganisations périodiques**.

→ réinsérer les données non effacées dans une nouvelle table

Chapitre 9 - Hachage Statique 29

5. Méthodes de résolution de collisions

3) Chaînage séparé

Les données en débordement (en cas de collisions) sont stockées dans un espace non «adressable» par la fonction de hachage. Par exemple à l'extérieur de la table sous forme de listes.

	donnée	vide	lien
0			
1	c	0	nil
2			
3	b	0	
4			
5	a	0	nil
6	d	0	
7			
N-1			

Le champ 'lien' lie les données en cases primaires avec leurs synonymes en débordement.

Le nombre de données insérées peut dépasser la taille de la table (N)

Chapitre 9 - Hachage Statique 30

5. Méthodes de résolution de collisions

3) Chaînage séparé

- Recherche de x

accès direct à la case $h(x)$

Si échec et non 'vide' Alors
Continuer la recherche en séquentielle dans la liste 'lien'

- Insertion de x

Si la case $h(x)$ est vide Alors
insertion dans cette case du tableau

Sinon
insertion dans la liste associée à cette case (en début de liste)

Chapitre 9 - Hachage Statique 31

5. Méthodes de résolution de collisions

3) Chaînage séparé: Algorithme d'insertion d'un élément

S1. [Hacher]
 $i := h(K)$

S2. [Existe-t-il une liste ?]
Si $T(i)$ est vide Alors ALLER A S5
{ dans les autres cas $T(i)$ est occupé; on consulte alors la liste des noeuds occupés }

$p := T(i)$

S3. [Comparer]
Si $K = \text{Donnée}(p)$ Alors l'algorithme se termine avec succès.

S4. [Avancer au prochain]
Si $\text{Lien}(p) \neq \text{Nil}$ Alors
 $p := \text{LIEN}(p)$
ALLER A S3

S5. [Insérer la nouvelle clé]
Allouer un maillon, soit q
 $\text{Donnée}(q) \leftarrow K$
 $\text{LIEN}(q) \leftarrow T(i)$
 $T(i) := q$

Chapitre 9 - Hachage Statique 32

5. Méthodes de résolution de collisions

3) Chaînage séparé

- Suppression x (physique)

Si x se trouve dans son adresse primaire (la donnée de la case $h(x)$) Alors

Si la liste 'lien' n'est pas vide Alors

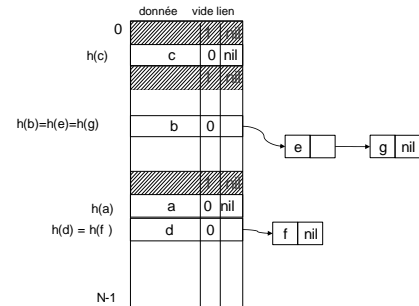
déplacer le premier élément de la liste dans la case $h(x)$ (en écrasant x)

Si la liste 'lien' est vide Alors
vider la case $h(x)$

Si x se trouve en débordement dans une liste Alors
la supprimer de cette liste

5. Méthodes de résolution de collisions

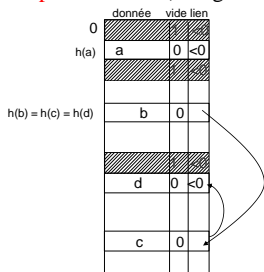
3) Chaînage séparé



5. Méthodes de résolution de collisions

4) Chaînage interne

Les données en débordement (en cas de collisions) sont stockés dans la table (**dans le même espace « adressable » par la fonction**) en gardant le chaînage entre les synonymes



Les 'liens' sont des entiers
(indices de cases)
<0 indique la fin de liste (nil)

5. Méthodes de résolution de collisions

4) Chaînage interne

- Recherche

Comme dans le chaînage séparé

- Insertion

Rechercher la donnée, si échec, 2 possibilités:

- 1) la case $h(x)$ est vide, on y insère la donnée
- 2) la case $h(x)$ n'est pas vide,

- soit i la dernière case visitée par la recherche

→ trouver une case vide

- insérer la donnée dans cette case vide et la chaîner à la suite de i

5. Méthodes de résolution de collisions

4) Chaînage interne: Algorithme d'insertion d'un élément

- Par convention $T(0)$ sera toujours vide.
- Une variable auxiliaire R est utilisée pour nous aider à déterminer les espaces vides. Quand la table est vide $R = N + 1$. Après plusieurs insertions on a : $T(j)$ occupé pour tout j tel que $R \leq j \leq N$.

- C1. [Hash]
 $i := h(K) + 1$ { donc $1 \leq i \leq N$ }
- C2. [Existe-t-il une liste ?]
 SI $T(i)$ est vide Alors ALLER A C6
- C3. [Comparer]
 SI $K = \text{Donnée}(i)$ Alors l'algorithme se termine avec succès.
- C4. [Avancer au prochain]
 SI $\text{LIEN}(i) < 0$ Alors
 $i := \text{LIEN}(i)$
 ALLER A C3

Chapitre 9 - Hachage Statique

37

5. Méthodes de résolution de collisions

4) Chaînage interne: Algorithme d'insertion d'un élément

C5. [Trouver le noeud vide]

{ la recherche est sans succès et nous voulons chercher une position vide dans la table }

Décrémenter R une ou plusieurs fois jusqu'à ce que $T(R)$ soit vide.

SI $R = 0$ l'algorithme se termine avec débordement.

Autrement faire :

$\text{LIEN}(i) := R$

$i := R$

C6. [Insérer la nouvelle clé]

Rendre $T(i)$ occupé avec

$\text{Donnée}(i) := K$

$\text{LIEN}(i) := 0$

Chapitre 9 - Hachage Statique

38

5. Méthodes de résolution de collisions

4) Chaînage interne

- Suppression (physique)

- Rechercher l'adresse i de la donnée (et de son prédécesseur i' dans la liste)
- Vérifier les suivants de i pour voir s'il y en a un qui a son adresse primaire = i auquel cas on le déplace vers i et on tente de vider son emplacement
- S'il n'y a pas de pb avec les suivants, on vide la case i en mettant à jour son prédécesseur i' pour « pointer » le suivant de i

→ PB: est-ce que le prédécesseur existe toujours ?

→ une solution: listes circulaires.

Chapitre 9 - Hachage Statique

39

6. Comparaison entre les différentes méthodes

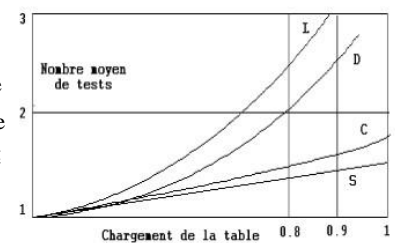
La figure suivante résume les courbes des nombres moyens de tests pour une recherche par rapport au chargement de la table (N/M , N étant le nombre des éléments présents dans la table) pour les quatre méthodes présentées.

L : essai linéaire

D : double hachage

C : chaînage interne

S : chaînage séparé



Chapitre 9 - Hachage Statique

40

6. Comparaison entre les différentes méthodes

Comme la figure le montre, les méthodes de chaînage semblent les meilleures. Cependant, leur inconvénient réside dans le champ additionnel représentant les liens.

Les méthodes de hachage ont de très bonnes performances.

Pour l'essai linéaire par exemple, même quand la table est remplie à 90%, le nombre de tests est au voisinage de 5.

Pour avoir un temps rapide (de l'ordre de 1), on convient de ne pas dépasser un chargement de 70%.

7. Etude de la distribution des données

Nous montrons dans cette étude que si la fonction de hachage est aléatoire, alors on peut prévoir le nombre de collisions, et donc comment les réduire.

- 1) Distribution de Poisson
- 2) Prévention des collisions
- 3) Réduction des collisions

7. Etude de la distribution des données

1) Distribution de Poisson

Supposons N adresses possibles et considérons les 2 événements :

A : une adresse donnée n'est pas choisie

B : une adresse donnée est choisie.

Cas où une donnée est hachée

Quand une donnée est hachée, l'un des événements (A ou B) se produit pour une adresse donnée.

Soit $P(A) = a$ et $P(B) = b$

$P(A)$ désigne la probabilité qu'une adresse donnée ne soit pas choisie.

$P(B)$ désigne la probabilité qu'une adresse donnée soit choisie.

On a donc :

$$P(B) = 1/N = b$$

$$P(A) = 1 - 1/N = (N-1)/N = a$$

Par exemple, si $N=10$ alors $a=0.9$ et $b=0.1$

7. Etude de la distribution des données

1) Distribution de Poisson

Cas où deux données hachent la même adresse

Qu'elle est la probabilité pour que deux données hachent la même adresse ?

$$P(BB) = b \cdot b = 1/N \cdot 1/N \text{ (événements indépendants)}$$

Qu'elle est la probabilité pour que la seconde donnée soit hachée dans une adresse différente de la première ?

$$P(BA) = b \cdot a = 1/N \cdot (N-1)/N$$

7. Etude de la distribution des données

1) Distribution de Poisson

Cas où deux données parmi quatre hachent la même adresse

Noter d'abord que $P(BABBA) = b.a.b.b.a = b^3a^2$.

Nous voulons connaître la probabilité pour qu'il existe un certain nombre de fois de B et de A (sans tenir compte de l'ordre). Par exemple, supposons que nous voulons hacher 4 données et que nous voulons savoir de quelles façons deux données hachent la même adresse.

On peut avoir les 6 événements suivants : AABB, BAAB, BABA, BBAA, ABBA, ABAB

$P = P(AABB) + P(BAAB) + \dots = 6 a^2b^2 = C_4^2 a^2b^2$.

C_4^2 représente le nombre de façons qu'on peut placer 2 A (et 2 B) dans 4 places.

Chapitre 9 - Hachage Statique 45

7. Etude de la distribution des données

1) Distribution de Poisson

Généralisation

Si x données parmi r hachent la même adresse on a x fois B et (r-x) fois A. La probabilité pour que x données parmi r hachent la même adresse est:

$$C_r^x \cdot a^{r-x} b^x \quad \text{avec} \quad C_r^x = r! / (x! (r-x)!)$$

Ce qui veut aussi dire :

Probabilité qu'une adresse donnée soit choisie x fois et ne soit pas choisie r-x fois.

Chapitre 9 - Hachage Statique 46

7. Etude de la distribution des données

1) Distribution de Poisson

Généralisation

Si N adresses possibles

$$P(x) = C_r^x \cdot a^{r-x} b^x$$

$$P(x) = C_r^x \cdot (1-1/N)^{r-x} (1/N)^x$$

P(x=0) veut dire probabilité pour qu'une adresse donnée ne soit jamais choisie.
 $P(0) = C_r^0 (1-1/N)^r (1/N)^0$

P(x=1) veut dire probabilité pour qu'une adresse donnée soit choisie une seule fois
 $P(1) = C_r^1 (1-1/N)^{r-1} (1/N)^1$

Chapitre 9 - Hachage Statique 47

7. Etude de la distribution des données

1) Distribution de Poisson

L'inconvénient de la formule est qu'elle est difficile à calculer pour N et r grands. La fonction de POISSON est une bonne approximation.

$$P(x) = (d^x \cdot e^{-d}) / x! \quad (\text{avec } d = r/N)$$

Si N est le nombre d'adresses possibles,
 r est le nombre de données insérées
 x est le nombre de données ayant la même adresse

Alors P(x) donne la probabilité que x données hachent la même adresse parmi r données insérées.

Par exemple

pour N=1000 adresses et r= 1000 données insérées on obtient les valeurs suivantes:
 $P(0) = .368$; $P(1) = .368$; $P(2) = .184$; $P(3) = .061$; etc.

Chapitre 9 - Hachage Statique 48

7. Etude de la distribution des données

1) Distribution de Poisson

En général, s'il existe N adresses, N.P(x) est le nombre de données qui hachent x fois la même adresse.

P(x) est aussi la proportion d'adresses ayant x données qui lui sont attribuées par hachage. Ceci nous permet donc de prévoir le nombre de collisions.

Chapitre 9 - Hachage Statique 49

7. Etude de la distribution des données

2) Prévention des collisions

Prenons N= 10 000 adresses possibles et r= 10 000 données insérées.

Quel est le nombre d'adresses qui n'ont **aucune** donnée attribuée ?
 $10\,000 * P(0) = 3679$

Quel est le nombre d'adresses qui n'ont qu'**une seule** donnée attribuée ?
 $10\,000 * P(1) = 3679$

Quel est le nombre d'adresses qui n'ont que **deux** données attribuées ?
 $10\,000 * P(2) = 1839$

Quel est le nombre d'adresses qui n'ont que **trois** données attribuées ?
 $10\,000 * P(3) = 613$

Pour 3679 données il n'y a pas de collisions.
 Pour 1839 données il y a collision (1839 seront en débordement)
 Pour 613 il y a collision (613 * 2 seront en débordement)

C'est une mauvaise répartition : nous avons des milliers d'adresses (3679) avec aucune donnée attribuée.

Chapitre 9 - Hachage Statique 50

7. Etude de la distribution des données

2) Prévention des collisions

On peut ainsi prévoir le nombre de données en débordement à l'avance. Il est donné par la formule

$$N (P(2) + 2P(3) + ...iP(i+1) +)$$

Chapitre 9 - Hachage Statique 51

7. Etude de la distribution des données

3) Réduction des collisions

Montrons à l'aide d'exemples d'une part, comment l'augmentation du nombre d'adresses possibles et d'autre part, comment l'utilisation des cases peuvent réduire les collisions.

Augmentation de l'espace des adresses

Nous définissons la densité d par : $d = r / N$
 où r est le nombre de données rangées
 et N est le nombre d'adresses possibles.

Regardons le comportement des fonctions de hachage (collisions) pour différentes valeurs de d.

$$P(x) = (d^x \cdot e^{-d}) / x! \quad (\text{avec } d = r/N)$$

P(x) dépend donc du rapport r/N c'est à dire de d.

Aussi, on constate un même comportement pour 500 données distribuées parmi 1000 adresses que 500.000 données distribuées parmi 1 million d'adresses. (d= 50% pour les deux cas)

Chapitre 9 - Hachage Statique 52

7. Etude de la distribution des données

3) Réduction des collisions

Prenons $d=0.5$ ($N=1000$ et $r=500$ données)

Combien d'adresses auront **0 donnée** attribuée ?

$$1\,000 * P(0) = 607$$

Combien d'adresses auront **1 donnée** attribuée ?

$$1\,000 * P(1) = 303$$

Combien d'adresses auront au moins **deux données** attribuées ?

$$1\,000 * (P(2) + P(3) + P(4) + \dots) = 90$$

avec $P(2)=0.758$; $P(3)=0.0126$; $P(4)=0.0016$; etc.

Quel est le nombre de données en **débordement** ?

$$1\,000 * (P(2) + 2 * P(3) + 3 * P(4) + 4 * P(5) + \dots) = 107$$

Quel est le pourcentage des données en débordement ?

$$107/500 = 21,4\%$$

7. Etude de la distribution des données

3) Réduction des collisions

On peut donc conclure :

Si la densité est de 50 %, on peut s'attendre à 79% de données rangées dans leur adresse primaire et 21 % rangées ailleurs.

7. Etude de la distribution des données

Utilisation des cases

On accepte b données par adresse possible.

Dans ce cas : $d = r / (b \cdot N)$

b : nombre de données par case

N : nombre de cases

r : nombre de données insérées.

	Sans case	Avec case
Nombre de donnée	$r=750$	$r=750$
Nombre d'adresses	$N=100$	$N=500$
Taille des cases	$b=1$	$b=2$
Densité	$d=0.75$	$d=0.75$
Rapport r/N	$r/N = 0.75$	$r/N = 1.5$

7. Etude de la distribution des données

Utilisation des cases

Nombre de données en débordement dans chaque cas :

$P(x)$	Sans case : $r/N = 0.75$	Avec case $r/N = 1.5$
$P(0)$	0.423	0.223
$P(1)$	0.354	0.335
$P(2)$	0.113 (Collisions)	0.251
$P(3)$	0.033 ""	0.126 (Collisions)
$P(4)$	0.006 ""	0.047 ""

Sans case ($b=1$) :

$$1\,000 \cdot [P(2) + 2 \cdot P(3) + 3 \cdot P(4) + \dots] \text{ débordements}$$

Avec case ($b=2$) :

$$500 \cdot [P(3) + 2 \cdot P(4) + 3 \cdot P(5) + \dots] \text{ débordements}$$

7. Etude de la distribution des données

Utilisation des cases

Ainsi, on conclut que:

l'utilisation des cases améliore les performances du hachage. Plus la case est grande, plus les performances sont meilleures.

7. Etude de la distribution des données

En résumé

Soit une table de N cases, et on aimerait insérer r données

Le pourcentage de remplissage (la densité) est donc: $d = r / N$

Soit $P(x)$ la probabilité que x données parmi r soient « hachées » vers la même case

$$P(x) = C_r^x (1 - 1/N)^{r-x} (1/N)^x$$

L'inconvénient de la formule est qu'elle est difficile à calculer pour N et r grands. La fonction de *POISSON* est une bonne approximation.

$$P(x) = (d^x * e^{-d}) / x! \quad (\text{avec } d = r/N)$$

$N * P(x)$: est donc une estimation du nombre de cases ayant été choisies x fois durant l'insertion des r données dans la table

Le nombre total de données en débordement est alors estimé à :

$$N(P(2) + 2 * P(3) + 3 * P(4) + 4 * P(5) + \dots)$$

7. Etude de la distribution des données

Exemple numérique:

Lors de l'insertion de $r=1000$ données dans une table de $N=1000$ cases

$$\text{densité} = d = r/N = 1$$

on estime que :

$N.P(0) = 368$	cases ne recevront aucune données
$N.P(1) = 368$	cases auront été choisies 1 seule fois
$N.P(2) = 184$	cases auront été choisies 2 fois
$N.P(3) = 61$	cases auront été choisies 3 fois
$N.P(4) = 15$	cases auront été choisies 4 fois
$N.P(5) = 3$	cases auront été choisies 5 fois
$N.P(6) = 0$	cases auront été choisies 6 fois

7. Etude de la distribution des données

Exemple numérique:

Avec la densité $d=1$ ($r = 1000$ et $N = 1000$)

Le nombre de données en débordement est proche de :

$$1000(0.184 + 2 * 0.061 + 3 * 0.015 + 4 * 0.003) = 363$$

soit **36,3%** des données

($r/\text{nombre de données en débordement} = 1000/363 = 36.3$)

contre 631 données dans leurs adresses primaires calculés:

$$1000(0.368 + 0.184 + 0.061 + 0.015 + 0.003) = 631$$

soit **63.1%** des données



8. Conclusion

Les méthodes de hachage donnent des résultats excellents en moyenne $O(1)$, mais lamentables dans le pire cas $O(n)$, car il n'est pas possible d'éviter les collisions.

En particulier, le choix de la fonction de hachage est fondamental.

Il existe plusieurs façons de réduire les collisions :

- trouver une fonction qui distribue bien les données c'est à dire de façon aléatoire.
- augmenter l'espace des adresses possibles.

Exemple: $d = 0.5$ ($N = 1000$ et $r = 500$ données)

Si la densité est de 50 %, on peut s'attendre à 79% de données rangées dans leur adresse primaire et 21 % rangées ailleurs.

- mettre plus d'une donnée par adresse possible (si on accepte par exemple b données par adresse donc $d = r/(b*N)$).