INF601 : Algorithme et Structure de données

Cours 5 : Table de hachage

B. Jacob

IC2/LIUM

9 mars 2010

Plan

Plan

Introduction

Table de hachage =

- Tableaux associatifs
- Hash tables

Accès aux éléments TDA précédents

Pour trouver la position d'un élément **e** dans un TDA de n éléments :

TDA liste : comparaison de la valeur des éléments de la liste avec **e**

$\longrightarrow$ au pire : comparaison jusqu'au dernier élément

$\longrightarrow$ recherche en $O(n)$

TDA arbre : comparaison de la valeur des éléments de l'arbre avec **e**

$\longrightarrow$ au pire : comparaison jusqu'à une feuille

$\longrightarrow$ recherche en $O(log(n))$

$\Rightarrow$ dépend du nombre d'éléments dans le TDA

$\Rightarrow$ si $n \nearrow$ alors temps de la recherche $\nearrow$

Avec une table de hachage :

Pour trouver la position d'un élément **e** dans une table de hachage de n éléments :

$\longrightarrow$ calcul de la position de **e** dans la table

$\longrightarrow$ accès direct à **e**

$\longrightarrow$ un seul accès pour accéder à **e**

$\longrightarrow$ recherche en $O(1)$

$\Rightarrow$ ne dépend pas de nombre d'éléments n
$\Rightarrow$ $\forall$ n temps de la recherche rapide
$\Rightarrow$ même si $n \nearrow$ alors temps de la recherche $= 1$

Principe

- la place d'un élément dans la table est calculée à partir de sa propre valeur
- calcul réalisé par une fonction de hachage : transforme la valeur de l'élément en une adresse dans un tableau
- recherche d'un élément : nombre constant de comparaisons $O(1)$. Ne dépend pas du nombre d'éléments dans le tableau

Plan

2 Utilisation d'une table de hachage

Fonction de hachage

Une table de hachage :

- utilisation d'une fonction de hachage
- l'indice d'un élément est donné par la fonction de hachage en fonction de la valeur de l'élément
- Pour une table T et un élément e
  $\exists$ une fonction de hachage h telle que $T[h(e)] = e$ (si $e \in T$)

Principe par l'exemple

Par exemple SI

- E
  - = Ensemble des éléments à stocker
  - = { serge, odile, luc, anne, annie, julie, basile, paula, marcel, elise }
- N
  - = Taille de la table
  - = 13

ALORS rôle de la fonction de hachage h
= associer à chaque élément e une position $h(e) \in [0..12]$

Fontion de hachage

Exemple d'algorithme de fonction h :

1. Attribuer aux lettres a,b,. . . ,z les valeurs 1,2,. . . ,26
2. $N \leftarrow \sum$ valeurs des lettres de e
3. $N \leftarrow N+$ nombre des lettres de e
4. $N \leftarrow mod(13)$

Calcul positions

La position de l'élément *serge* est donnée par $h(serge)$

$\rightarrow$ $h(serge) = (54 + 5) mod 13 = 7$

$\rightarrow$ *serge* est à la position 7 dans la table de hachage

De même :

- $h(odile) = (45 + 5) mod 13 = 11$
- $h(luc) = (36 + 3) mod 13 = 0$
- $h(anne) = (34 + 4) mod 13 = 12$
- $h(annie) = (43 + 5) mod 13 = 9$
- $h(jean) = 8$, $h(julie) = 10$, $h(basile) = 2$, $h(paule) = 4$, $h(elise) = 3$, $h(marcel) = 6$

Tableau de hachage

0	luc
1	
2	basile
3	elise
4	paula
5	
6	marcel
7	serge
8	jean
9	annie
10	julie
11	odile
12	anne

Tableau de hachage

serge ?

0	luc
1	
2	basile
3	elise
4	paula
5	
6	marcel
7	serge
8	jean
9	annie
10	julie
11	odile
12	anne

Tableau de hachage

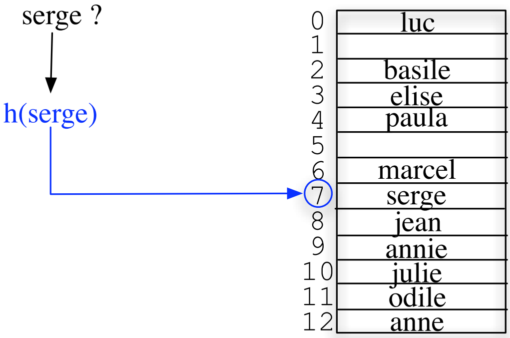
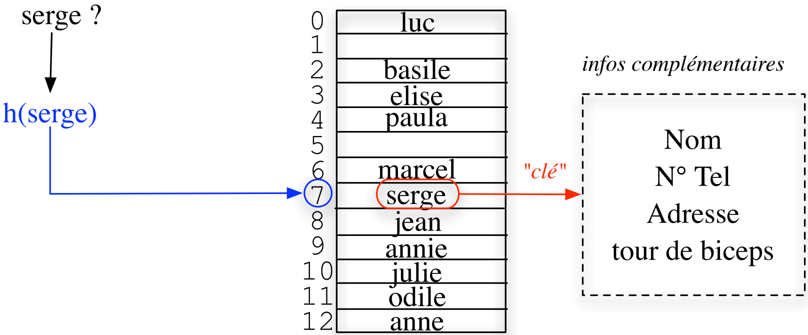


Tableau de hachage



Plan

3. Méthodes de hachage
   - Méthode par Extraction
   - Méthode par Compression
   - Méthode par Division
   - Méthode par Multiplication

Rôle de la fonction

Une méthode de hachage

- transforme la valeur d'un élément en position
- doit être déterministe
  (pour retrouver les éléments c'est mieux)
- doit être facilement calculable
  (temps d'exécution de la fonction rapide sinon on perd le bénéfice de l'accès en O(1))

Choix de la fonction

Avoir une "bonne" fonction de hachage
$\longrightarrow$ dépend de l'ensemble des éléments sur lequel on travaille

Exemple :

- éléments = chaines de caractères
- fonction = codes binaires des 2 premières lettres

Cette fonction est

$\longrightarrow$ OK pour les prénoms

$\longrightarrow$ KO pour les noms de fonction des TDA en C
(commencent tous par les mêmes lettres, le nom du TDA)

Principe de hachage

Principe montré sur un exemple

Base : dans la suite les éléments seront

- éléments :
  - des mots (chaines de caractères)
  - leur valeur est une suite de bits formées par la concaténation des codes binaires de leurs lettres sur 5 bits
- fonction = construction d'un indice à partir de cette suite de bits

Exemples de valeurs d'éléments

Code binaires des lettres :

A = 00001	F = 00110	K = 01011	P = 10000	U = 10101
B = 00010	G = 00111	L = 01100	Q = 10001	V = 10110
C = 00011	H = 01000	M = 01101	R = 10010	W = 10111
D = 00100	I = 01001	N = 01110	S = 10011	X = 11000
E = 00101	J = 01010	O = 01111	T = 10100	Y = 11001

Z = 11010

Valeurs :

- mot "LES" = 01100 00101 10011
- mot "CAR" = 00011 00001 10010

But : construire une fonction h telle que
$h(mot) \rightarrow$ indice dans $[0..T-1]$ (T taille de la table)

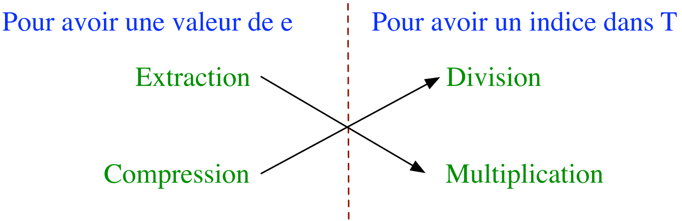
Méthodes de hachage

Présentation de quelques principes de construction de fonctions de hachage qui permettent :

$\longrightarrow$ de dégager quelques techniques utiles

$\longrightarrow$ d'éviter les pièges les plus courants

Principales méthodes :



Méthodes pour la valeur

Le but est de transformer la valeur de l'élément en une valeur "représentable"

- un mot mémoire
- la place mémoire d'un entier ...

Présentation de 2 méthodes :

- par extraction
- par compression

Plan

3. Méthodes de hachage
   - Méthode par Extraction
   - Méthode par Compression
   - Méthode par Division
   - Méthode par Multiplication

Principe

$\longrightarrow$ On extrait une partie de la valeur (de la chaîne de bits) de l'élément e

Par exemple

- seulement certains bits
- seulement certaines zones de bits

Exemple

extraction des bits 1,2,7 et 8 de l'élément

élément	représentation	h(élément)$_2$	h(élément)$_{10}$
ET	0010110100	1000	8
OU	0111110101	1101	13
NI	0111001001	1101	13
IL	0100101100	0000	0

Avantages/Inconvénients

Avantages :

- calcul facile à mettre en oeuvre
- si $N = 2^p - 1$ alors h $\longrightarrow$ un indice dans T

Inconvénients :

adaptée seulement à des cas particuliers

- quand on connaît la valeur des éléments *a priori*
- quand on sait que certains bits ne sont pas significatifs

En général, méthode par extraction $\longrightarrow$ pas de bons résultats car ne dépend pas de la totalité de la valeur de l'élément.

Règle :

Une bonne fonction de hachage utilise toute la valeur de e

Plan

Principe

→ On utilise tous les bits de e pour calculer son indice dans la table.

Par exemple

1. on découpe la chaîne de bits de l'élément en morceaux d'égale longueur
2. on additionne les morceaux. Pour éviter les débordements on peut utiliser, à la place de l'addition, l'opération booléenne "ou exclusif" (`xor`)

Exemple

Elément	Calcul	h(élément)$_2$	h(élément)$_{10}$
ET	00101 `xor` 10100	10001	17
OU	01111 `xor` 10101	11010	26
NI	01110 `xor` 01001	00111	7
CAR	00011 `xor` 00001 `xor` 10010	10000	16

Inconvénient

Problème :

- Hache de la même façon toutes les permutations d'un même mot
  $\longrightarrow$ h(CAR) = h(ARC)
- Vient du fait que toutes les sous chaines de bits sont des représentations de caractères
  $\longrightarrow$ plusieurs sous-chaines peuvent être identiques

Règle : une "bonne" fonction de hachage doit briser les sous-chaînes de bits

Solution

Par exemple décaler circulairement les sous-chaînes de bits

- 1^{iere} sous-chaîne $\longrightarrow$ 1 bit vers la droite
- 2^{ieme} sous-chaîne $\longrightarrow$ 2 bits vers la droite
- 3^{ieme} sous-chaîne $\longrightarrow$ 3 bits vers la droite...

Dans ce cas

Elément	Calcul	$h(\text{élément})_2$	$h(\text{élément})_{10}$
CAR	10001 `xor` 01000 `xor` 01010	10011	19
ARC	10000 `xor` 10100 `xor` 01100	01000	8

Avantages :

$\longrightarrow$ valeur de e = taille d'un mot mémoire

$\longrightarrow$ combinaison possible avec les méthodes qui suivent

Méthodes pour l'indice

Le but est de ramener la représentation de e dans à indice de la table T.

- indice $\in [0..T-1]$
- indice $\in [1..T]$

Présentation de 2 méthodes :

- par division
- par multiplication

Plan

Principe

$\rightarrow$ On calcule le reste de la division de la valeur de e par N, la taille de la table.

$$h(e) = e \bmod N$$

Exemple

Taille de la table $N = 37$

Elément	$(\text{Elément})_{10}$	Calcul	$h(\text{élément})_{10}$
ET	180	$180 \bmod 37$	32
OU	501	$501 \bmod 37$	20

Avantages/Inconvénients

Avantages :

- facile et rapide à calculer

Inconvénients :

- dépend trop de N
  Par exemple
  - si N est pair alors tous les éléments vont aller dans les indices pairs de T
  - si N est impair alors tous les éléments vont aller dans les indices impairs de T
  - si N a des petits diviseurs...

$\longrightarrow$ h n'est pas uniforme

$\longrightarrow$ il faudrait que la taille N de T soit un nombre premier (comme dans l'exemple) ... mais il peut tout de même y avoir des phénomènes d'accumulation

Plan

3. Méthodes de hachage
    - Méthode par Extraction
    - Méthode par Compression
    - Méthode par Division
    - Méthode par Multiplication

Principe

$\rightarrow$ Basé sur la multiplication de e par un nombre réel θ $(0 < \theta < 1)$

Algorithme :

1. $r \leftarrow e * \theta$
2. on garde la partie décimale de r
3. $r \leftarrow r \times N$: on multiplie par la taille du tableau
4. on garde la partie entière de r

$$h(e) = [((e \times \theta) \bmod 1) \times N]$$

Exemple

Supposons que :

- $\theta = 0.6125423371$
- $N = 30$

Alors h(ET)

$$
\begin{aligned}
&= [((180 \times \theta) \bmod 1) \times N] \\
&= [(110.87016302 \bmod 1) \times 30] \\
&= [0.87016302 \times 30] \\
&= 26
\end{aligned}
$$

Avantages/Inconvénients

Avantages :

- la taille du tableau est sans importance

Inconvénients :

- la valeur de θ doit être choisie avec soin
  $\longrightarrow$ pas trop près de 0 ni de 1 pour éviter les accumulations aux extrémités de la table

Des études théoriques montrent que les valeurs de θ qui répartissent uniformément les éléments sont :

$\theta = (\sqrt{5} - 1)/2 \approx 0.6180339887$

$\theta = 1 - (\sqrt{5} - 1)/2 \approx 0.3819660113$

Conclusion sur les méthodes de hachage

- pas de fonction de hachage universelle
- une "bonne" fonction doit être
  - rapide à calculer
  - répartir uniformément les éléments dans T
- mais cela dépend
  - de la machine
  - de l'application (contenu/valeur des éléments)

Limites de la fonction de hachage

But de la fonction h : attribuer 1 élément à chaque position de T
$\longrightarrow$ pas toujours possible.

- il peut y avoir des positions de T qui ne sont pas utilisées
  $\Rightarrow$ pb de "gaspillage mémoire"
  $\Rightarrow$ pb de taux de remplissage trop bas

- il y peut y avoir une place de T qui est attribuée à plusieurs éléments
  $\Rightarrow$ pb des collisions

Plan

4. Taux de remplissage d'une table

Calcul du taux

bon taux de remplissage $\Leftrightarrow$ nb de positions vides $\searrow$

$$taux\ de\ remplissage = \frac{nombre\ de\ positions\ occupees}{nombre\ total\ de\ positions}$$

avec nombre total de positions

$=$ taille de la table

$=$ T

But :

- avoir un taux le plus proche de 1
  $\longrightarrow$ toutes les positions de T sont remplies
- avoir le moins de collisions possible
  $\longrightarrow$ qu'il n'y ait qu'un élément par position

Plan

5. Résolutions des collisions
   - Méthodes indirectes
   - Méthodes directes

Introduction

Rappel : collisions = plusieurs éléments à la même position

On montre qu'il est pratiquement impossible , même pour la meilleure des fonctions de hachage d'éviter les collisions. Il est donc nécessaire de savoir les résoudre.

2 méthodes de résolution :

- résolution par chaînage : les éléments qui ont la même position sont chaînés entre eux, à l'extérieur ou à l'intérieur de T
- résolution par calcul : lorsqu'il y a collision, on calcule à partir de l'élément une nouvelle place dans T

Plan

Types de méthodes indirectes

Rappel : hachage indirect $=$ les éléments en collision sont chaînés entre eux

2 types de gestion des collisions indirectes :

1. Par hachage avec chaînage séparé
2. Par hachage coalescent

Hachage avec chaînage séparé

Tous les éléments en collision sont chaînés entre eux à l'extérieur de la table de hachage T

- les éléments de T sont alors des LISTES
- recherche, ajout, suppression d'un élément en collision *idem* TDA LISTE
- la liste doit elle être triée ?

Hachage avec chaînage séparé

Tous les éléments en collision sont chaînés entre eux à l'extérieur de la table de hachage T

- les éléments de T sont alors des LISTES
- recherche, ajout, suppression d'un élément en collision *idem* TDA LISTE
- la liste doit elle être triée ?
  SI fonction hachage "bonne" ALORS nb collisions $\searrow$
  $\longrightarrow$ recherche séquentielle dans liste non triée suffit

Hachage avec chaînage séparé

Tous les éléments en collision sont chaînés entre eux à l'extérieur de la table de hachage T

- les éléments de T sont alors des LISTES
- recherche, ajout, suppression d'un élément en collision *idem* TDA LISTE
- la liste doit elle être triée ?
  SI fonction hachage "bonne" ALORS nb collisions $\searrow$
  $\longrightarrow$ recherche séquentielle dans liste non triée suffit
  SINON (nb collisions $\nearrow$) ?

Hachage avec chaînage séparé

Tous les éléments en collision sont chaînés entre eux à l'extérieur de la table de hachage T

- les éléments de T sont alors des LISTES
- recherche, ajout, suppression d'un élément en collision *idem* TDA LISTE
- la liste doit elle être triée ?
  SI fonction hachage "bonne" ALORS nb collisions $\searrow$
  $\longrightarrow$ recherche séquentielle dans liste non triée suffit
  SINON (nb collisions $\nearrow$) ?
  $\longrightarrow$ Liste triée (+ algo de recherche adapté)

Hachage avec chaînage séparé

Tous les éléments en collision sont chaînés entre eux à l'extérieur de la table de hachage T

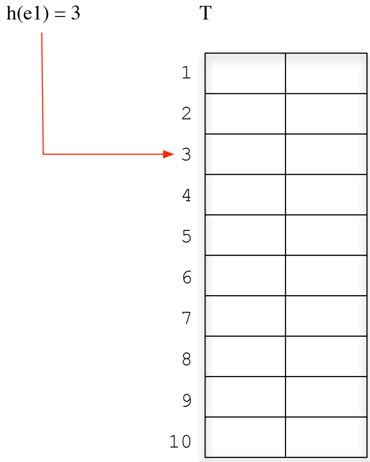
- les éléments de T sont alors des LISTES
- recherche, ajout, suppression d'un élément en collision *idem* TDA LISTE
- la liste doit elle être triée ?
  SI fonction hachage "bonne" ALORS nb collisions $\searrow$
  $\longrightarrow$ recherche séquentielle dans liste non triée suffit
  SINON (nb collisions $\nearrow$) ?
  $\longrightarrow$ Liste triée (+ algo de recherche adapté)
  $\longrightarrow$ Arbre binaire de recherche (équilibré)

Exemple chaînage séparé

T

1		
2		
3		
4		
5		
6		
7		
8		
9		
10		

Exemple chaînage séparé

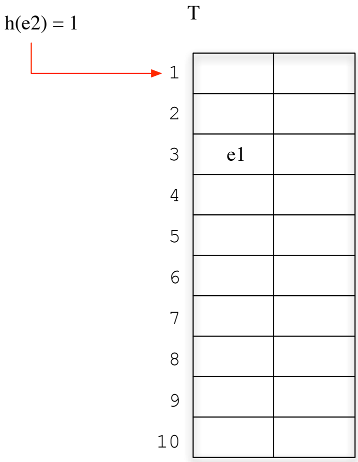


Exemple chaînage séparé

T

1		
2		
3	e1	
4		
5		
6		
7		
8		
9		
10		

Exemple chaînage séparé

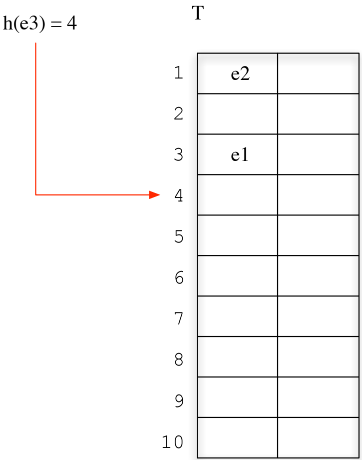


Exemple chaînage séparé

T

1	e2	
2		
3	e1	
4		
5		
6		
7		
8		
9		
10		

Exemple chaînage séparé

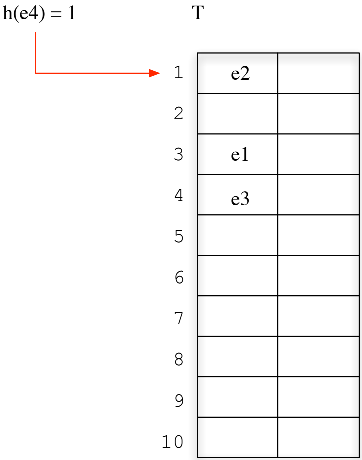


Exemple chaînage séparé

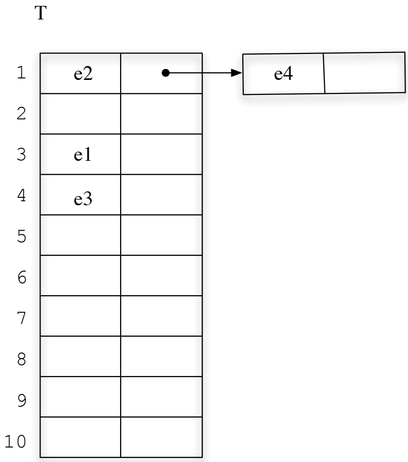
T

1	e2	
2		
3	e1	
4	e3	
5		
6		
7		
8		
9		
10		

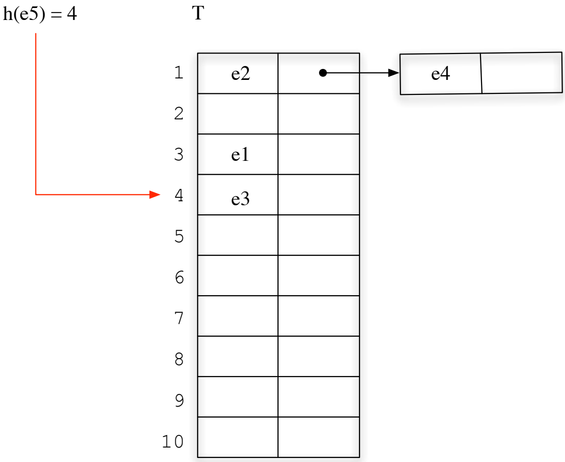
Exemple chaînage séparé



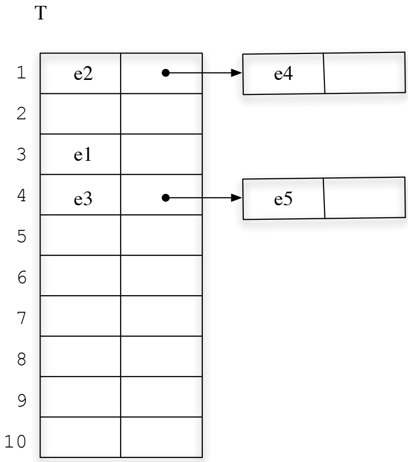
Exemple chaînage séparé



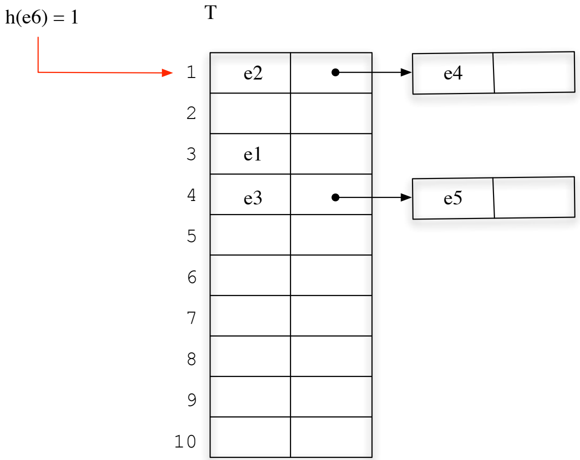
Exemple chaînage séparé



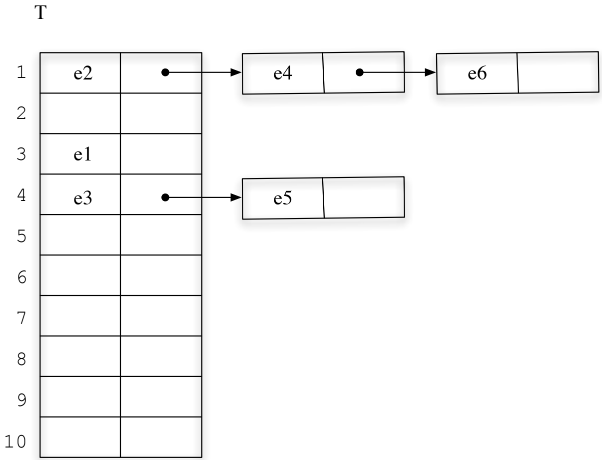
Exemple chaînage séparé



Exemple chaînage séparé



Exemple chaînage séparé



Hachage coalescent

Tous les éléments en collision sont chaînés entre eux à l'intérieur de la table de hachage T

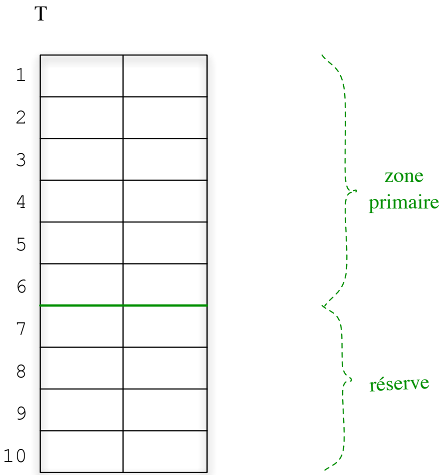
- pourquoi ? chaînage séparé $\Rightarrow$ allocation dynamique de la mémoire : pas toujours possible
- réserver alors *a priori* tout l'espace mémoire nécessaire (positions "normales" + positions collisions)
- la taille N de T est donc fixe

Dans ce cas on divise T en 2 zones :

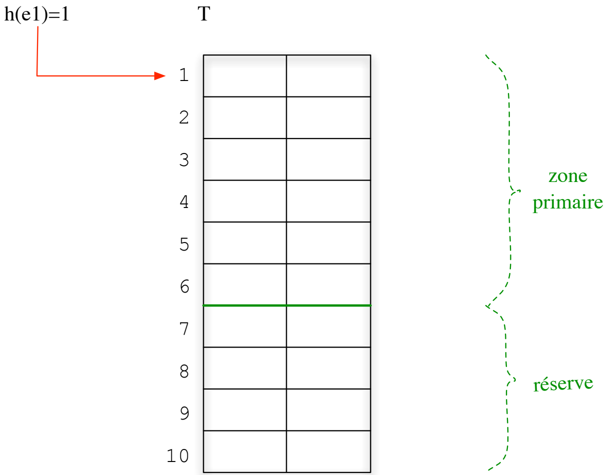
1. une zone d'adresses primaires de capacités p pour les positions "normales"
2. une zone d'adresses de réserve de capacités r pour les collisions

Les valeurs p et r sont fixées elles aussi *a priori*

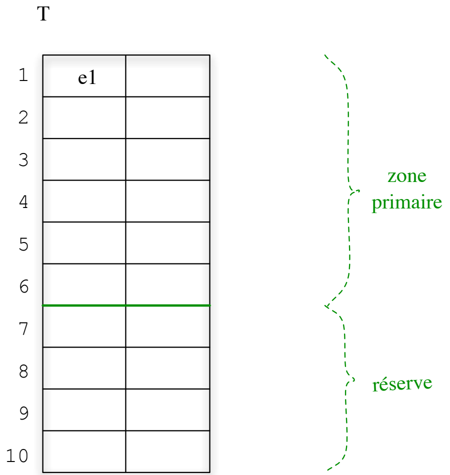
Exemple chaînage coalescent



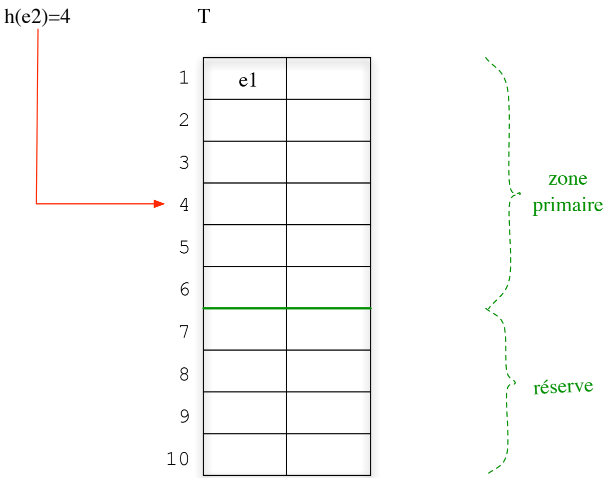
Exemple chaînage coalescent



Exemple chaînage coalescent



Exemple chaînage coalescent



Exemple chaînage coalescent

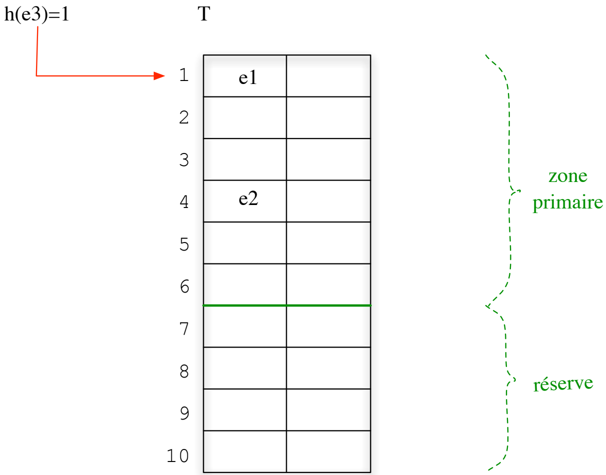
T

1	e1	
2		
3		
4	e2	
5		
6		
7		
8		
9		
10		

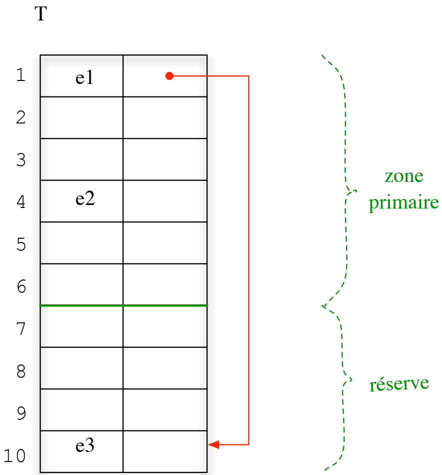
zone primaire

réserve

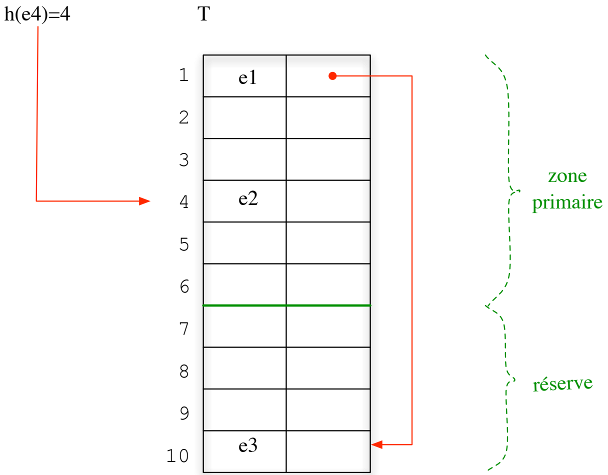
Exemple chaînage coalescent



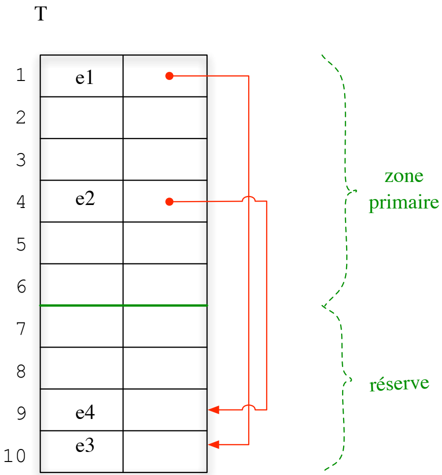
Exemple chaînage coalescent



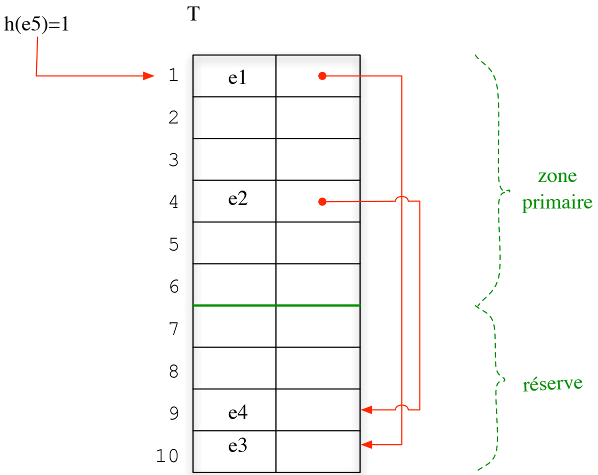
Exemple chaînage coalescent



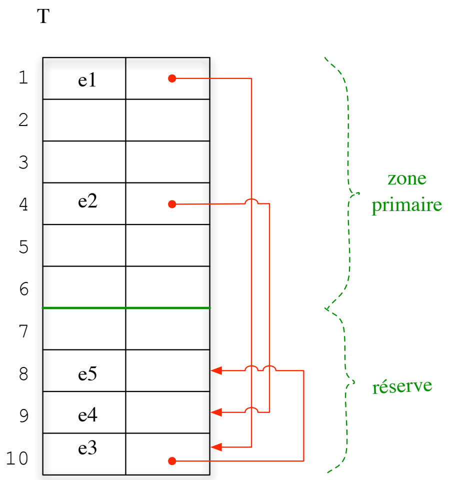
Exemple chaînage coalescent



Exemple chaînage coalescent



Exemple chaînage coalescent



Fusion des zones avec hachage coalescent

On peut penser à fusionner la zone primaire et la réserve

MAIS

- mélanges entre
  - les éléments qui sont en collision et chaînés entre eux
  - les éléments primaires

  $\longrightarrow$ hachage avec coalescence de listes (ou hachage coalescent)
- coalescences des listes $\Rightarrow$ nouvelles collisions : les collisions secondaires
- temps de recherche alors plus long que le chaînage en dehors de T

Suppression avec chaînage coalescent

Supprimer un élément dans T :

- problème plus compliqué que l'adjonction
- il faut décaler les éléments pour mettre à jour un chaînage dans T
  $\longrightarrow$ plutôt que supprimer physiquement un élément,
  on préfère souvent marquer sa place comme vide
  (pas de chaînage à modifier)

Plan

Types de méthodes directes

Rappel : hachage direct $=$ les éléments en collision sont mis à de nouvelles positions dans T. Ces nouvelles positions sont gérées par calcul

2 types de gestion des collisions directes :

1. Par hachage linéaire
2. Par double hachage

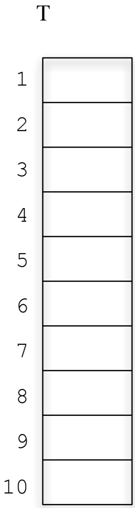
Hachage linéaire

Si collision à la position i → on essaie $i + 1 \bmod N$

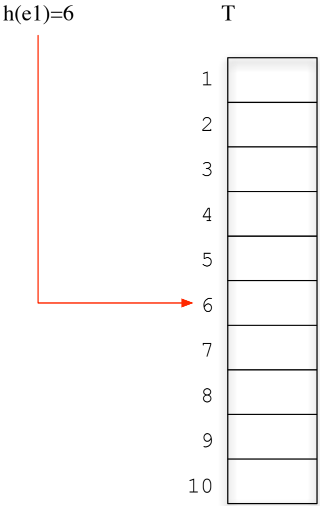
Algorithme d'ajout d'un élément :

```
i <-- h(e)
SI i occupée ALORS
   j <-- 1
   i <-- (h(e)+j) mod N
   TQ ((i occupée) ET (j<T-1)) FRE
      j <-- j + 1
      i <-- (h(e)+j) mod N
   FTQ
FSI
```

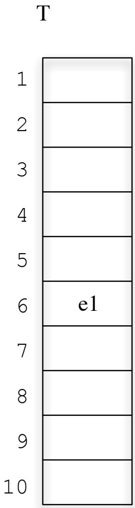
Exemple chaînage linéaire



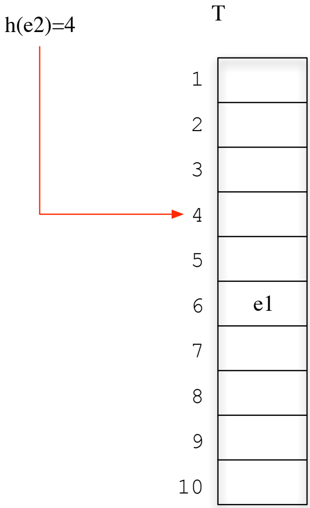
Exemple chaînage linéaire



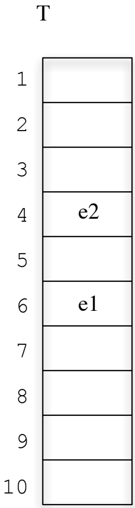
Exemple chaînage linéaire



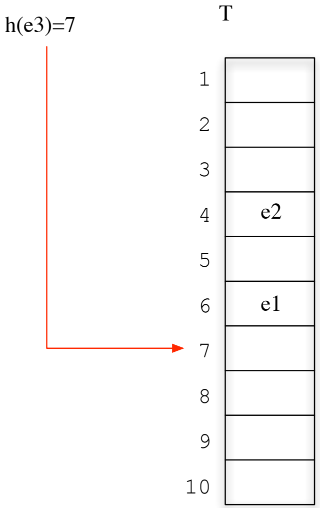
Exemple chaînage linéaire



Exemple chaînage linéaire



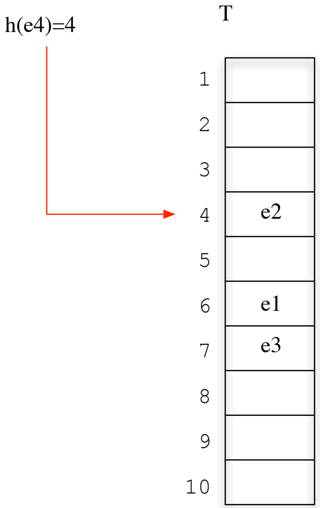
Exemple chaînage linéaire



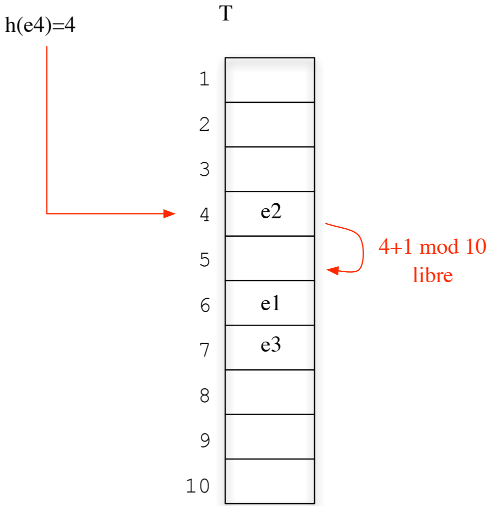
Exemple chaînage linéaire

	T
1	
2	
3	
4	e2
5	
6	e1
7	e3
8	
9	
10	

Exemple chaînage linéaire



Exemple chaînage linéaire

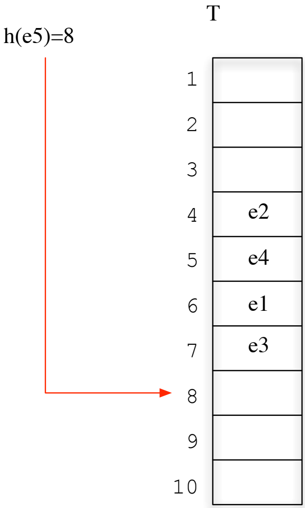


Exemple chaînage linéaire

T

1	
2	
3	
4	e2
5	e4
6	e1
7	e3
8	
9	
10	

Exemple chaînage linéaire

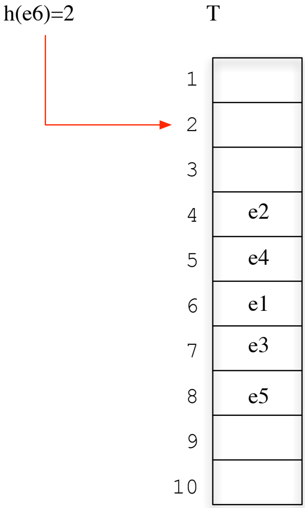


Exemple chaînage linéaire

T

1	
2	
3	
4	e2
5	e4
6	e1
7	e3
8	e5
9	
10	

Exemple chaînage linéaire

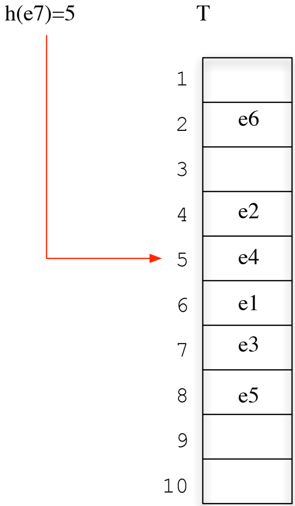


Exemple chaînage linéaire

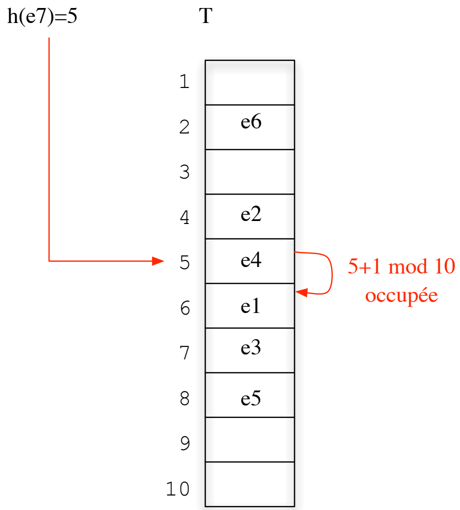
T

	T
1	
2	e6
3	
4	e2
5	e4
6	e1
7	e3
8	e5
9	
10	

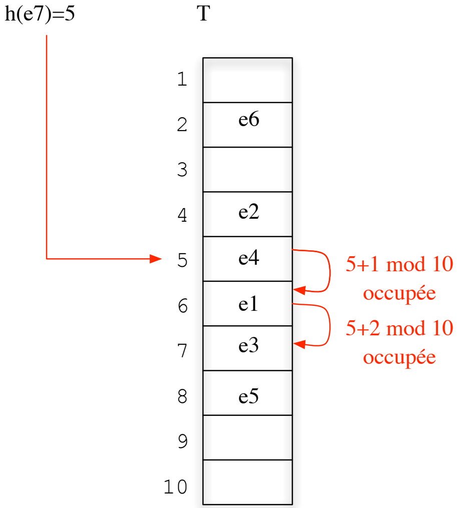
Exemple chaînage linéaire



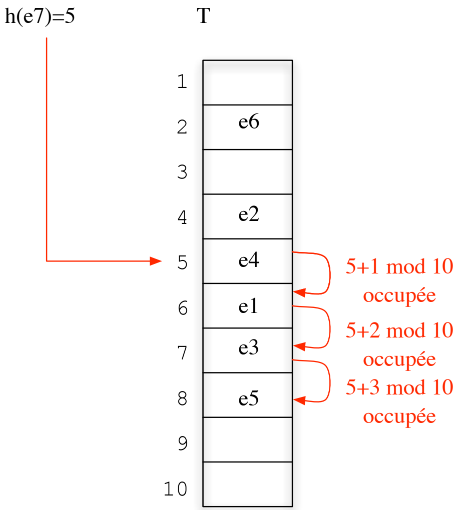
Exemple chaînage linéaire



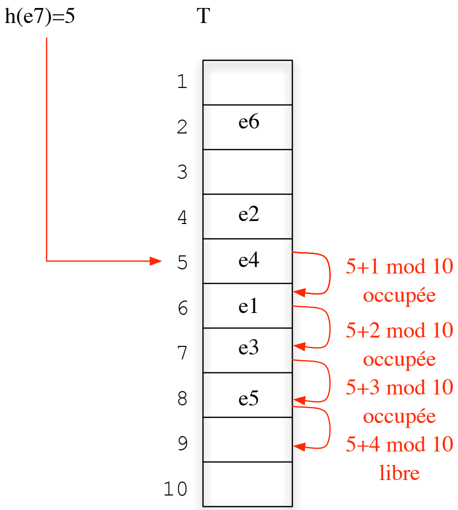
Exemple chaînage linéaire



Exemple chaînage linéaire



Exemple chaînage linéaire



Exemple chaînage linéaire

T

1	
2	e6
3	
4	e2
5	e4
6	e1
7	e3
8	e5
9	e7
10	

Recherche chaînage linéaire

Algorithme de recherche d'un élément *e* dans *T*

```
i <-- h(e)
j <-- 1
TQ (T[i] != e) ET (T[i] != vide) ET (j < N-1) FRE
    i <-- (h(e)+j) mod N
FTQ
SI T[i] == e ALORS
    --> trouve
SINON
    --> pas trouve
FSI
```

Double hachage

Si collision à la position $i \rightarrow$ on essaie $k \times i + 1 \bmod N$
(avec k nombre fixé)

Ceci pour éviter les regroupements aux alentours de $i + 1 \bmod N$

MAIS regroupement quand même

- $\rightarrow$ il faut que k dépende de e
- $\rightarrow$ calcul de k par une seconde fonction de hachage $h'(e)$ (d'où le double hachage)
- $\rightarrow$ si N premier, il suffit que $h'(e) \rightarrow [1..N-1]$
- $\rightarrow$ si $N = 2^p$, il suffit que $h'(e) \rightarrow$ indice impair

Double hachage

Algorithme :

```
i <-- h(e)
SI i occupée ALORS
   j <-- 1
   i <-- (h(e) + h'(e) * j) mod N
   TQ ((i occupée) ET (j<T-1)) FRE
      j <-- j + 1
      i <-- (h(e) + h'(e) * j) mod N
   FTQ
FSI
```

Bibliographie

- TYPE DE DONNÉES ET ALGORITHMES

  Auteurs : Marie-Claude Gaudel, Michèle Soria, Christine Froidevaux

  Volume : Vol. II "Recherche, Tri, Algorithmes sur les graphes"

  Editeur : Collection Didactique, Editions INRIA

  ISBN : 2-7261-0490-8

That all folks...