

Compactage

L'utilisation de listes chaînées est assez gourmande en mémoire, et nécessite des allocations dynamiques coûteuses en temps

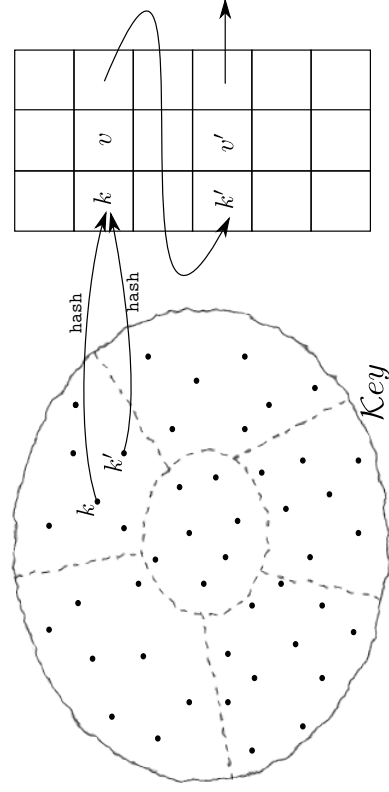
Si on a n éléments dans une table de hachage de taille n , on aimerait ne pas avoir à utiliser plus de n fois la taille d'une clef et de la valeur correspondante.

- ▶ On utilise le tableau lui même pour stocker les associations qui seraient à la suite dans la liste chaînée

On parle alors de **table de hachage fermée**

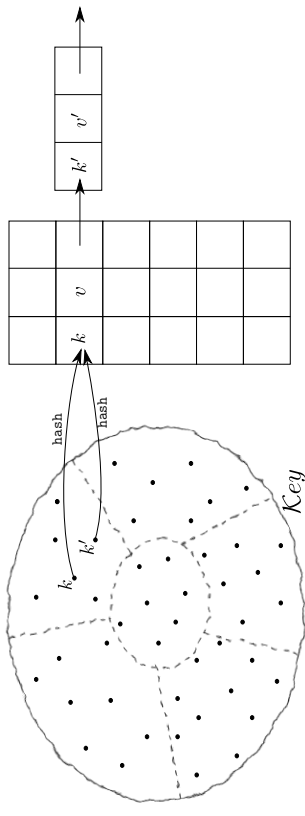
2^e idée

On utilise le tableau pour les éléments suivants de la liste chaînée



1^{re} idée

Le tableau ne contient pas des pointeurs vers des listes chaînées, mais les premiers éléments de la liste

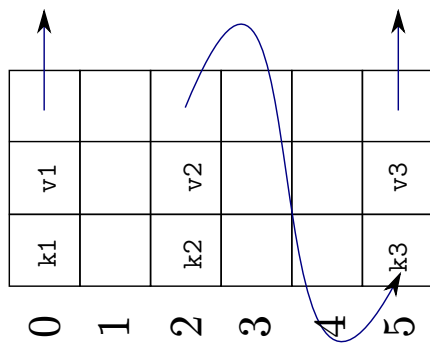


Rechercher : exemple 1

rechercher (d, k)

- ▶ $\text{hash}(k) = 1$

Pas trouvée

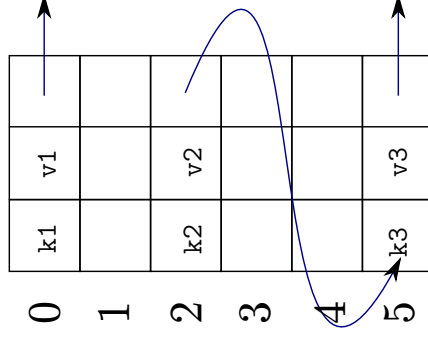


Rechercher : exemple 2

rechercher(d, k_3)

- ▶ $\text{hash}(k_3) = 2$

On suit les liens jusqu'à trouver la
clef (ou non)

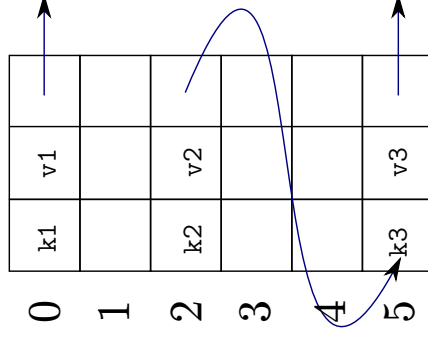


Rechercher : exemple 3

rechercher(d, k')

- ▶ $\text{hash}(k') = 5 \neq \text{hash}(k_3)$

Pas trouvée



Orbite

Deux associations (k, v) et (k', v')

sont dans la même **orbite** si

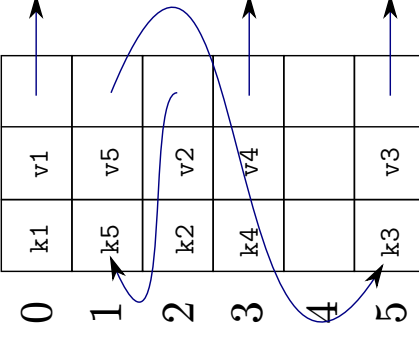
$\text{hash}(k) = \text{hash}(k')$

Orbites :

$\{(k_1, v_1)\}$

$\{(k_2, v_2); (k_5, v_5); (k_3, v_3)\}$

$\{(k_4, v_4)\}$



Recherche

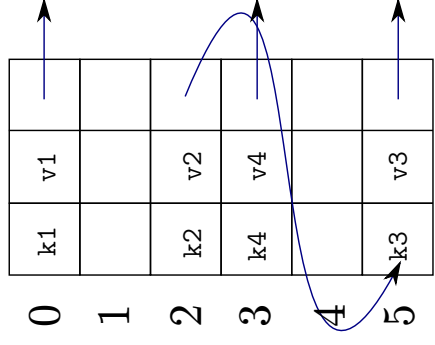
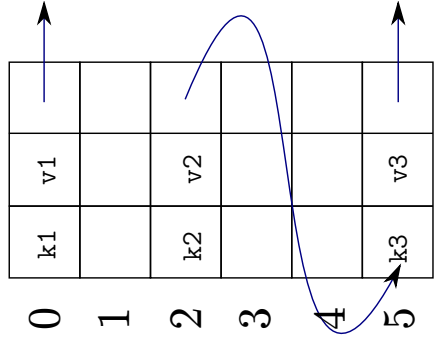
rechercher(d, k)

- ▶ on calcule $\text{hash}(k)$
- ▶ si la case en $\text{hash}(k)$ est vide
 - clef non trouvée
- ▶ si la case en $\text{hash}(k)$ contient une clef de hachage différent de $\text{hash}(k)$
 - clef non trouvée
 - ▶ sinon
 - si la clef est k , on renvoie la valeur
 - sinon on va à la case indiquée dans le successeur

Insertion : exemple 1

`insérer(d, k4, v4)`

► $\text{hash}(k4) = 3$



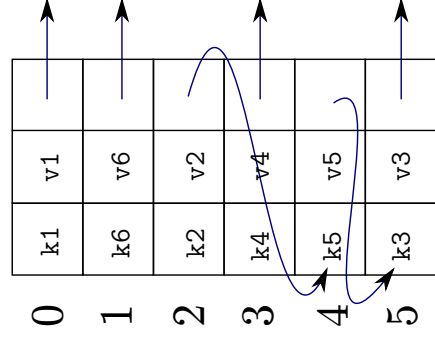
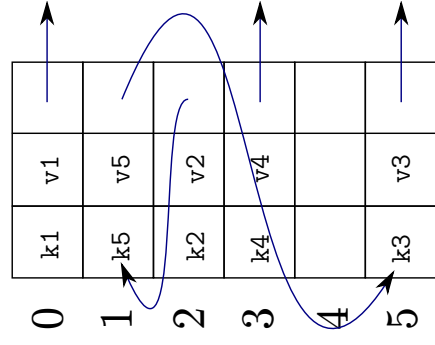
ENSIE : Programmation avancée

28/37

Insertion : exemple 3

`insérer(d, k6, v6)`

► $\text{hash}(k6) = 1$



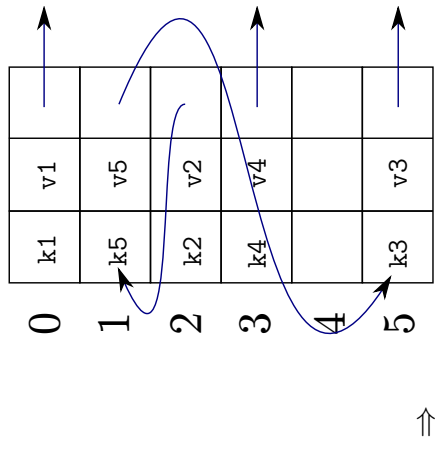
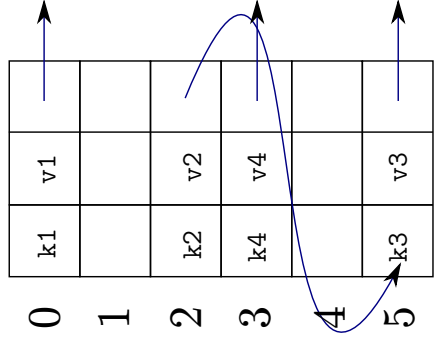
ENSIE : Programmation avancée

30/37

Insertion : exemple 2

`insérer(d, k5, v5)`

► $\text{hash}(k5) = 2$



ENSIE : Programmation avancée

29/37

Insertion

`insérer(d, k, v)`

- on calcule $\text{hash}(k)$
- si la case $\text{hash}(k)$ est libre, on y met k, v
- si la case $\text{hash}(k)$ est occupée par une clef k' avec $\text{hash}(k') = \text{hash}(k)$
 - on trouve une case vide à la position j
 - on remplit la case j avec k, v
 - on met comme successeur en j le successeur en $\text{hash}(k)$
 - on met j comme successeur en $h(k)$

ENSIE : Programmation avancée

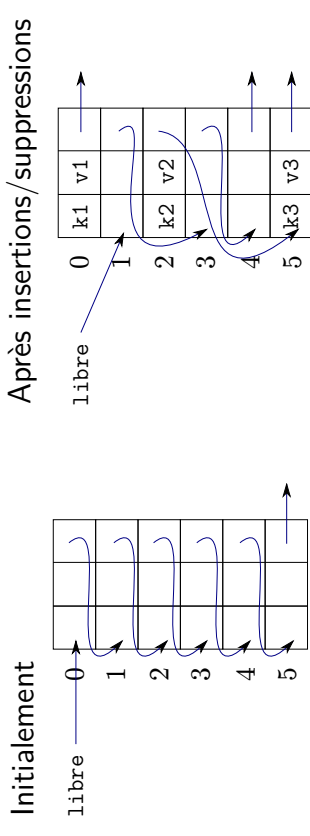
31/37

Insertion (suite)

- ▶ sinon on a k', v', s' en $h(k)$ avec $hash(k') \neq hash(k)$
 - on met k, v en $hash(k)$
 - on suit les successeurs à partir de $hash(k')$ jusqu'au prédécesseur j de $hash(k)$
 - on met k', v', s' dans une case libre
 - on met à jour le successeur de j sur cette case

Gestion des cases libres

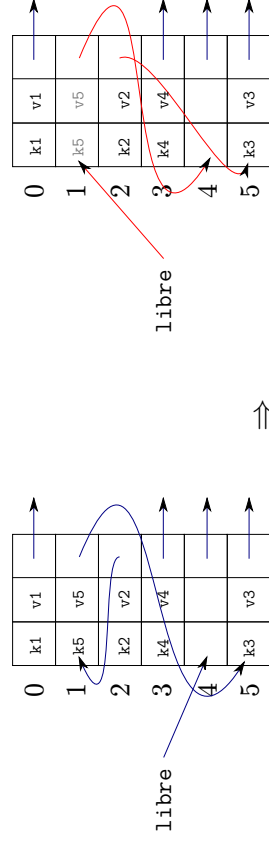
On a un pointeur vers la première case libre, et on utilise le successeur pour avoir les suivantes



Suppression : exemple

supprimer($d, k5$)

- ▶ $hash(k5) = 2$



Suppression

supprimer(d, k)

- ▶ on trouve k en i
- ▶ on met à jour le prédécesseur éventuel en j :
 - on met comme successeur en j le successeur en i
 - ▶ on libère la case correspondante :
 - on fait pointer Libre vers i
 - on met comme successeur en i l'ancienne valeur de Libre

Module paramétré

En OCaml, on a réutilisé les listes d'association pour faire les tables de hachage

- ▶ on ne peut utiliser que les fonctions déclarées dans l'interface de `Liste_assoc`
- ▶ on aurait en fait pu utiliser n'importe quel dictionnaire

Foncteur : module qui prend en paramètre une interface de module
 \simeq fonction sur les modules