

Algorithmique et structures de données

Chapitre 10 – Les tables de symboles et tables de hachage

Samar MOUCHAWRAB, PhD Eng

2A Cycle Préparatoire – Semestre 1
2017/2018

Introduction

Un **dictionnaire** est une structure de données gardant en mémoire des entrées de la forme (clé, élément).

Le but d'un dictionnaire est d'être capable d'accéder rapidement à une entrée, en donnant la valeur de sa clé.

Ici, contrairement aux files de priorité, on n'a pas besoin d'une relation d'ordre pour comparer les clés.

Un dictionnaire ayant une relation d'ordre pour comparer ses clés est appelé un dictionnaire ordonné.

Par contre, on doit pouvoir décider si deux clés sont égales. On utilisera pour cela un testeur d'égalité.

Exemples d'application

Comptes de banque:

Clé: Numéro du compte

Élément: un dossier contenant l'information sur le propriétaire du compte, une liste de transactions (retraits, dépôts)...

Dictionnaire interactif:

Clé: mot cherché

Élément: la définition de ce mot

Testeur d'égalité

Un testeur d'égalité est un type abstrait de données qui teste l'égalité de deux clés.

Quand un dictionnaire doit savoir si deux clés sont égales, il utilise un testeur d'égalité.

Opérations:

égal(a,b): Retourne vrai si a et b sont considérés égaux, et faux sinon.
Retourne une erreur si a et b ne peuvent être comparés.

Map ou table associative

Une “**map**” ou **table associative**, est un dictionnaire dans lequel l’insertion de plusieurs éléments ayant la même clé est interdite.

Les opérations principales sur les “maps” sont celles de recherche, d’insertion et de suppression d’éléments = (clé, élément)

Exemples d’applications:

- Carnet d’adresses
- Base de données des relevés de notes des étudiants

Tables de symboles

Les dictionnaires et les maps sont tous les deux des types de données abstraits appelés les tables de symboles. Le but du jeu est en fait de stocker des paires "clé-valeur" et de pouvoir y accéder efficacement, en utilisant seulement la clé.

Dictionnaire ou Map ??

- Le répertoire téléphonique est un dictionnaire puisqu'on peut avoir deux contacts qui portent le même nom, mais qui ont des numéros différents. Ou bien on peut associer à une même personne plusieurs numéros, par exemple son téléphone fixe et son téléphone portable.
- Un dictionnaire est également un dictionnaire puisque certains mots peuvent avoir plusieurs significations.
- A l'opposé, l'unicité des id des news d'un site est primordiale pour savoir laquelle consulter.
- Dans le cas de PIDs (Process ID), il est primordial d'avoir des PIDs uniques pour savoir quel processus terminer.

Opérations sur une map

Opérations principales:

Recherche par clé : Si la “map” M a une entrée de clé k, retourne la valeur associée à cette entrée, sinon retourne NULL.

Insertion d'une valeur avec une clé : insérer l'entrée (k,e) dans M. Ajoute la clé et la valeur dans la collection en retournant la valeur insérée. Si la clé existe déjà, sa valeur sera écrasée par celle passée en paramètre .

Supprimer une valeur avec une clé : Si la “map” M a une entrée de clé k, l'enlever de M et retourner l'élément associé, sinon retourner NULL

Opérations auxiliaires:

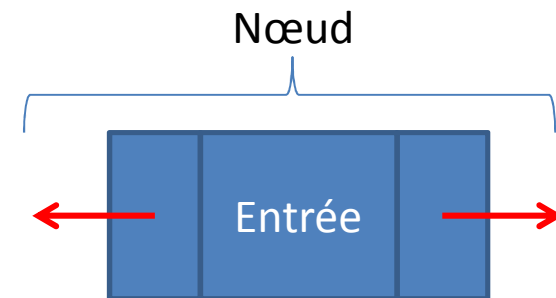
taille(): Retourne le nombre d'entrées dans M.

estVide(): Retourne VRAI si M est vide, et FAUX sinon.

Implémentation d'une map

On peut implémenter une “map” comme une liste non ordonnée, doublement chaînée. On garde en mémoire la liste avec les entrées dans un ordre arbitraire.

```
typedef struct Entree Entree;
struct Entree {
    int cle;
    int valeur;
};
typedef struct Noeud Noeud;
struct Noeud {
    Entree entree;
    Noeud *suivant;
    Noeud *precedent;
};
typedef struct Map Map;
struct Map{
    Noeud *tete;
};
```



Opérations sur un dictionnaire

Opérations principales:

Recherche par clé : Si le dictionnaire a une entrée de clé k , retourne la valeur associée à cette entrée, sinon retourne NULL.

Chercher tout par clé : Retourne un itérateur de toutes les entrées de clé k , ou NULL si aucune entrée de clé k .

Insertion d'une valeur avec une clé : insérer l'entrée (k,e) dans le dictionnaire.

Supprimer une valeur avec une clé : Si le dictionnaire a une entrée de clé k , l'enlever et retourner sa valeur associée, sinon retourner NULL

Opérations auxiliaires:

taille(): Retourne le nombre d'entrées dans M .

estVide(): Retourne VRAI si M est vide et FAUX, sinon.

Implémentation d'un dictionnaire

On peut implémenter un dictionnaire avec une liste non ordonnée, doublement chaînée comme dans le cas d'un map.

Lorsque les clés des entrées sont des entiers, on peut facilement implémenter le dictionnaire à l'aide d'un tableau de longueur N . La cellule d'indice i du tableau est vue comme un contenant pour les entrées de clé i .

Si aucune entrée du dictionnaire a une clé k , la cellule k contient un élément spécial `No_Key`. Si on essaie d'insérer plus d'un élément de clé k , il y a collision.

L'implémentation avec un tableau engendre les problèmes suivants:

- Les clés doivent être des entiers positifs entre 0 et $N-1$ (où N est la longueur de la table)
- La complexité est d'ordre N , ce qui est souvent beaucoup plus grand que n , le nombre d'entrées du dictionnaire.

Implémentation d'un dictionnaire

Il y a d'autres structures de données qui peuvent être utilisées pour implémenter les dictionnaires comme:

- Listes triées
- Arbres
- Tables de Hachage

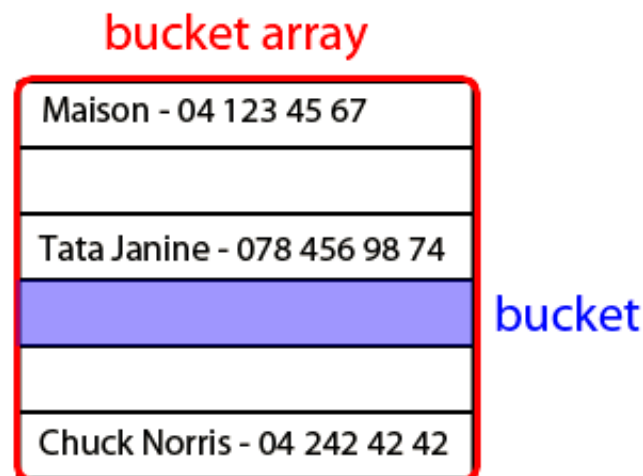
Dans la suite du chapitre nous allons donner plus de détails sur les tables de hachage.

Tables de Hachage

Le mécanisme de table de hachage est simple : les paires "clé-valeur" sont placées dans un tableau de manière efficace. En fait, au lieu de les ajouter l'une après l'autre comme on aurait tendance à faire dans un tableau, on les éparpille le plus uniformément possible.

Le tableau porte généralement le nom de **bucket array** tandis que les cases, vides ou occupées, sont les **buckets**.

Les indices de chaque paire sont déterminés par le **hach code** de la clé, qui est donné par une **fonction de hachage**.



Fonction de hachage

Une fonction de hachage h transforme les clés d'un certain type en des entiers entre 0 et $N-1$.

Exemple: $h(x) = x \bmod N$ est une fonction de hachage lorsque les clés sont des entiers.

Une fonction de hachage est dite **parfaite** si elle transforme des clés **différentes en entiers différents**.

Une table de hachage pour un type de clés donnés, consiste en :

- Une fonction de hachage h
- Un tableau de taille N

Lorsqu'on implémente un dictionnaire à l'aide d'une table de hachage, le but est d'insérer l'entrée (k, v) dans la cellule d'indice $i = h(k)$

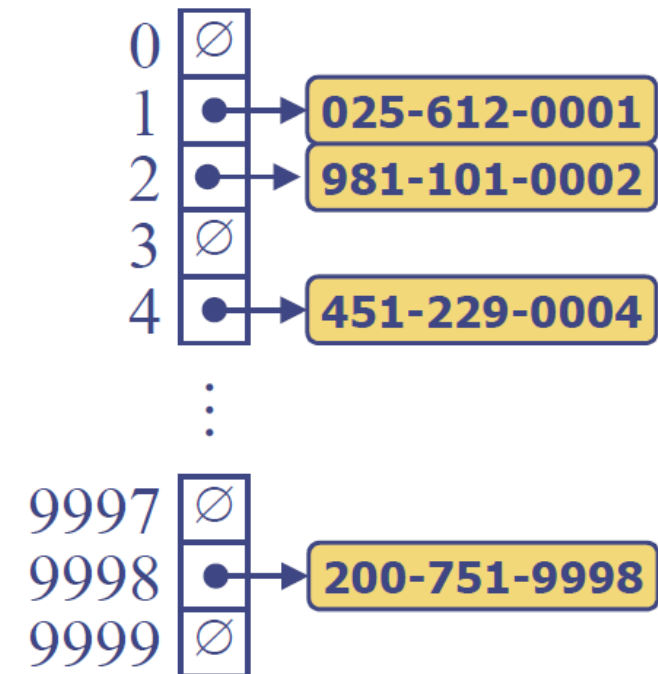
Exemple : extraction

On veut implémenter un dictionnaire dont les entrées sont de la forme (numéro, NOM), où numéro est un code d'employés de 9 chiffres.

Exemple : 020-111-6314

Notre table de hachage utilise un tableau de taille $N = 10\,000$ et la fonction de hachage est :

$h(x)$ = les 4 derniers chiffres de x



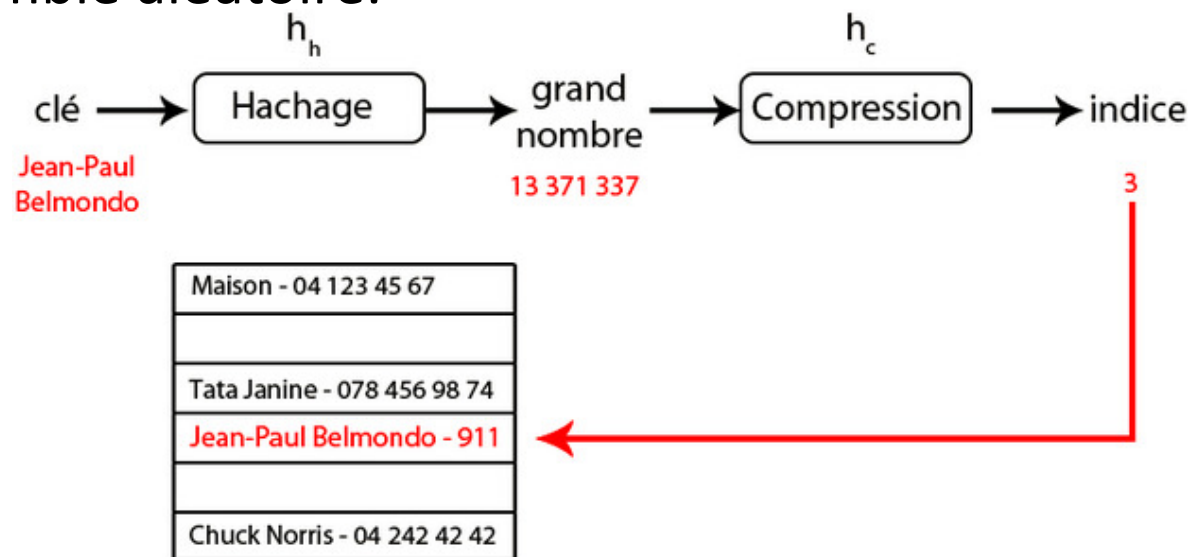
Fonctions de hachage

Une fonction de hachage est définie habituellement comme la composition de deux fonctions:

1. *Le code de hachage* : $h_h : \text{clé} \rightarrow \text{entiers}$
2. *La fonction de compression* : $h_c : \text{entiers} \rightarrow [0, N-1]$

Le code de hachage est utilisé en premier, puis la fonction de compression: $h(x) = h_c(h_h(x))$

Le but de la fonction de hachage est de « disperser » les clés d'une façon qui semble aléatoire.



Codes de hachage

Illustrons le mécanisme de code de hachage avec un petit exemple numérique. Imaginons que l'on dispose des fonctions suivantes :

$$h_h(i) = 3*i + 14$$

$$h_c(i) = i \bmod 10$$

On aimerait insérer les clés suivantes dans un tableau de 10 cases (les valeurs associées n'ont ici pas beaucoup d'importance, on va les laisser de côté) : 0, 4, 7 et 42 . On obtient les résultats ci-dessous :

Clé	h_h	h_c
0	14	4
4	26	6
7	35	5
42	140	0

si on place les clés dans un tableau, on aura

42				0	7	4			
----	--	--	--	---	---	---	--	--	--

Gestion des collisions

Un problème de taille peut survenir lorsque deux clés génèrent le même indice dans la table de hachage.

Reprenons l'exemple, et imaginons que l'on veut insérer la clé 22. On calcule : $h_h(22) = 3 * i + 14 = 80$ et $h_c(80) = 0$
Or l'indice 0 est déjà occupé par la clé 42

Lorsque deux clés différentes sont envoyées sur la même case, on dit qu'il y a **collision**, et il va falloir les gérer car plus le tableau sera rempli plus il y aura de chances d'en avoir une.

Plusieurs méthodes existent. On peut en citer:

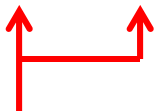
- le sondage
- le double hachage
- le chaînage linéaire.

Le sondage linéaire

Le sondage linéaire est appelé **probing** en anglais. Cette méthode est la plus intuitive : si on a une collision, à partir de l'indice, on parcourt le tableau jusqu'à trouver une case libre.


L'insertion

indices	0	1	2	3	4	5	6	7	8	9
clés	42	22			0	7	4			



Si on veut insérer la clé 30, $h_h(30) = 3 * i + 14 = 104$ et $h_c(104) = 4$
La case 4 est occupée, on parcourt alors les indices jusqu'à trouver une case libre. La première case libre est en 7, on y met **30** !

indices	0	1	2	3	4	5	6	7	8	9
clés	42	22			0	7	4	30		



Le sondage linéaire

La recherche

Pour rechercher une clé, c'est le même principe : on calcule son indice, puis on avance dans le tableau jusqu'à trouver la clé. Si on arrive sur une case vide, c'est que la clé n'existe pas (sinon on l'aurait rencontrée en chemin).

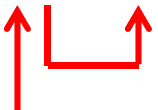
Le sondage linéaire

La suppression

La suppression est plus délicate que l'insertion car elle peut rendre inefficace la recherche. Si la suppression rend une case vide sur le "chemin" à parcourir, alors on perdra l'accès à certaines clés !

Par exemple, imaginons qu'on supprime purement et simplement la clé **7** qui est à l'indice 5, après avoir inséré la clé **30**.

indices	0	1	2	3	4	5	6	7	8	9
clés	42	22			0		4	30		



Dès lors, on ne saura jamais trouver **30** vu que le "chemin" pour y accéder passe par une case vide, à laquelle le programme va s'arrêter.


Le sondage linéaire

La suppression

Une solution simple pour éviter ces pertes est, lorsqu'on demande la suppression d'une clé, de la garder mais de marquer la case comme libre.

Ainsi, lorsqu'on cherchera **30**, on passera dessus sans problèmes. Il faut juste penser, dans la méthode pour ajouter une paire "clé-valeur", à **chercher la première case vide ou marquée libre**.

indices	0	1	2	3	4	5	6	7	8	9
clés	42	22			0	7	4	30		



Le sondage linéaire

Avantages et inconvénients

Le système de sondage est simple, ne requiert pas d'espace mémoire en plus, mais malheureusement souffre d'un problème de clustering.

En fait, les clés ont tendance à s'agglutiner, à former des grappes (clusters) dans le tableau. Le souci arrive quand il faudra parcourir toute la grappe pour trouver une case disponible ou une clé.

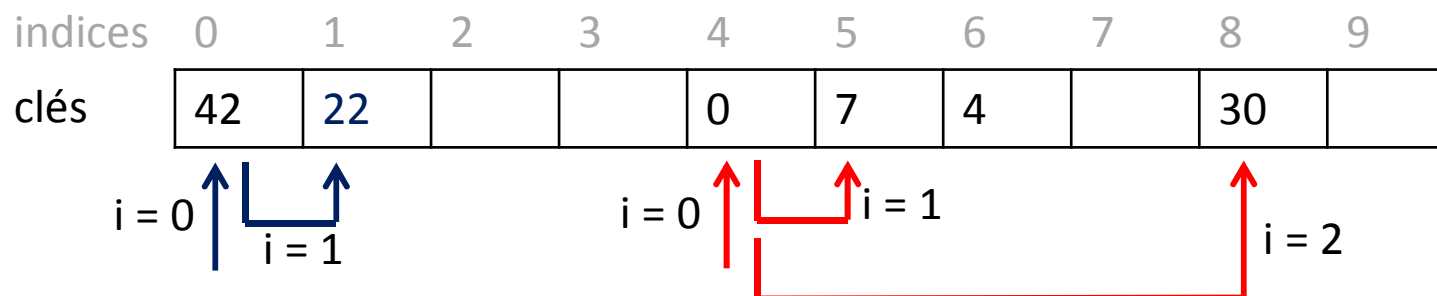
Si on reprend le même exemple, si une nouvelle clé tombe à l'indice 4, il va falloir faire 4 tests pour trouver sa place définitive. Dans ce cas-ci c'est peu, mais avec des tableaux de centaines de milliers, voir de millions d'éléments, la performance de l'algorithme baisse. Une manière d'éviter ces grappes est d'utiliser un sondage d'un degré plus élevé. Des clusters existeront toujours, mais seront plus petits.

Le sondage quadratique

Un moyen d'éviter le *clustering* du sondage linéaire est d'utiliser un sondage quadratique : le principe est exactement le même, c'est la forme des indices qui change.

$$index = (h(k) + c_1 i + c_2 i^2) \bmod M \quad (M \text{ est la taille de la table})$$

Considérons $c_1=0$ et $c_2=1$ dans l'exemple précédent. Si on insère **22** (dont le hach code est 0), on testera les indices 0 et 1. Pour insérer **30**, on ira d'abord voir en 4, puis en 5 et enfin en 8.



Le double hachage

Le principe du **double hachage** est similaire à celui du sondage. *On cherche la première case disponible en faisant varier un indice i , mais en plus de l'incrémenter, on va aussi le hacher. Ainsi, cela évitera du *clustering*.*

Si on prend comme seconde fonction de hachage h_2

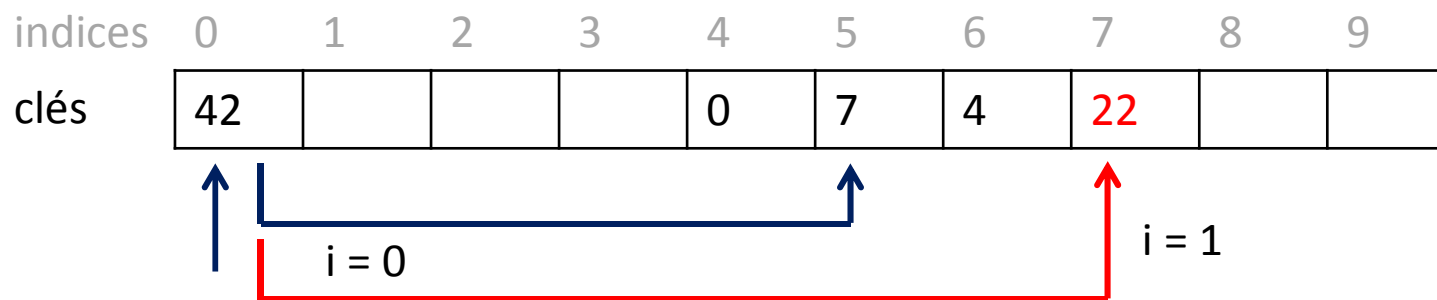
$$indice = (h(k) + h_2(i)) \bmod M \quad (M \text{ est la taille de la table})$$

Complétons l'exemple précédent avec $h_2(i) = 2*i + 5$

Si on veut insérer **22**. Pour rappel, $h(22) = 0$

Pour $i = 0$, $h_2(0) = 5$ qui est occupé

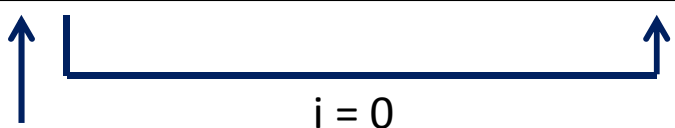
Pour $i = 1$, $h_2(1) = 7$ qui est disponible



Le double hachage

Pour insérer **30**, son hach code compressé est 4. On a ainsi:
Pour $i = 0$, $h_2(0) = 5$, indice = $4 + 5 = 9$ qui est disponible

indices	0	1	2	3	4	5	6	7	8	9
clés	42				0	7	4	22		30



$i = 0$

Le double hachage

Avantages et inconvénients:

Grâce à ce second hachage, on va parcourir le tableau dans tous les sens, cela évite le phénomène de *clustering*.

En revanche, cela peut amener un autre problème : si la seconde fonction de hachage est mal choisie, il se peut qu'il y aura beaucoup d'incrémentations de i pour trouver une case libre ! Cela peut dans certains cas durer plus longtemps que le sondage linéaire. Même, on pourrait ne jamais trouver de case libre, même s'il y en a dans le tableau !

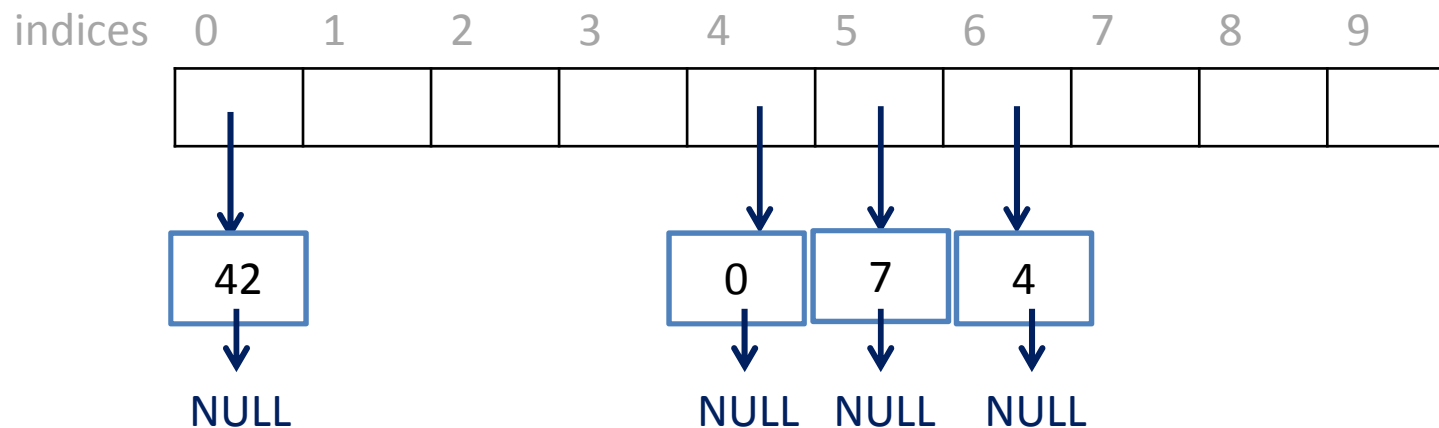
Il est possible de choisir une fonction de double hachage qui permettra de vérifier au moins une fois toutes les cases du tableau, mais gardez à l'esprit que le processus peut être long !

En résumé, comme le sondage, le double hachage devient moins performant lorsque le tableau est fort rempli, mais avec un remplissage raisonnable et si la fonction est bien choisie, il évite les *clusters* et est très efficace.

Le chaînage linéaire

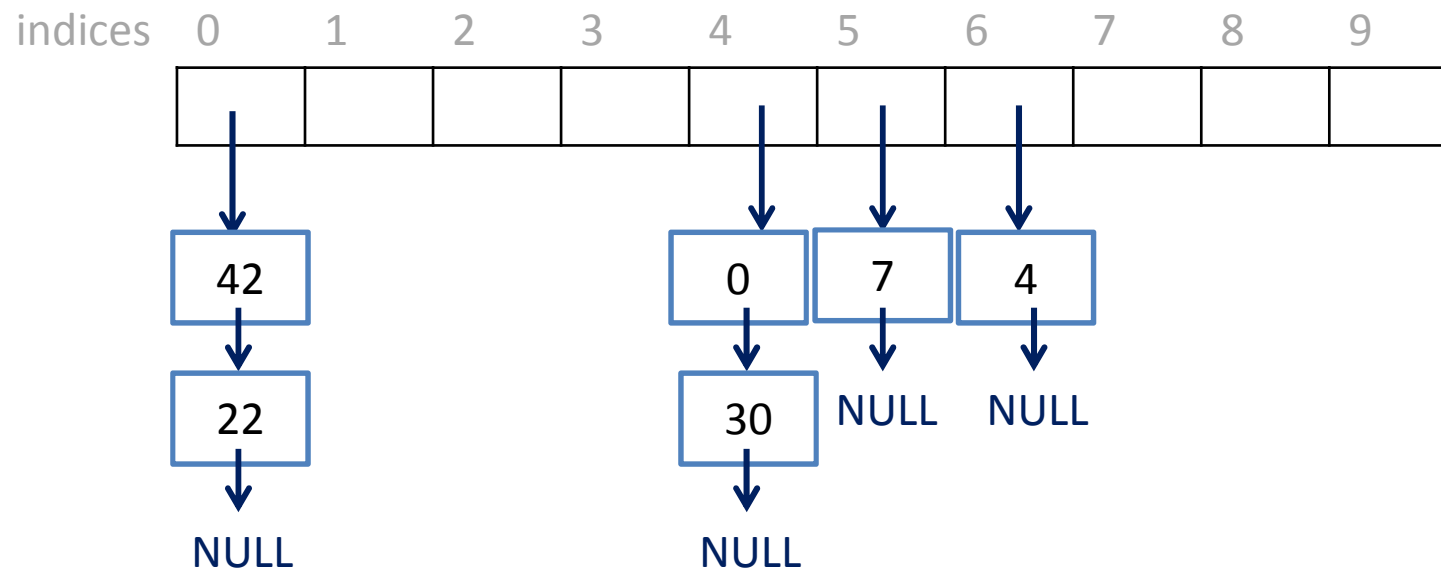
La méthode est simple. Au lieu d'avoir un élément unique dans le bucket, on aura une liste.

Lorsqu'on place un élément dans le *bucket*, on l'ajoute dans la liste, et quand il y a une collision, tout simplement on l'ajoute à la liste.



Le chaînage linéaire

Insérer 22 et 30 donne :



Le chaînage linéaire

Avantages et inconvénients :

Contrairement aux sondage et double hachage, on utilise dans le chaînage linéaire des structures externes au tableau, en l'occurrence des files.

Autant ce désavantage en espace mémoire est comblé dans des situations où les deux autres méthodes dégénèrent (avec des tableaux bien remplis), autant ces dernières sont à préférer pour un remplissage du tableau modéré.

De plus, si la fonction de hachage n'est pas optimale, il est possible que de longues files se créent, ce qui inévitablement va détériorer les performances.