Dictionnaires : tables de hachage

Fabrice Lembrez - PSI*

Lycée Pierre de Fermat

Comment les dictionnaires sont-ils implémentés ?
Cela aidera à introduire les bases de données.

Plan

Dictionnaire = table de hachage

Le dictionnaire c'est le type concret, la table de hachage le modèle abstrait.

Rappel du principe : association clé $\rightarrow$ valeur

- avec un choix arbitraire, non fixé, de clés

- et pourtant la recherche de la présence d'une clé en $O(1)$.

Dictionnaire = table de hachage

Le dictionnaire c'est le type concret, la table de hachage le modèle abstrait.

Rappel du principe : association clé $\rightarrow$ valeur

- avec un choix arbitraire, non fixé, de clés

- et pourtant la recherche de la présence d'une clé en $O(1)$.

Problématique des tables de hachage : suivre la clef

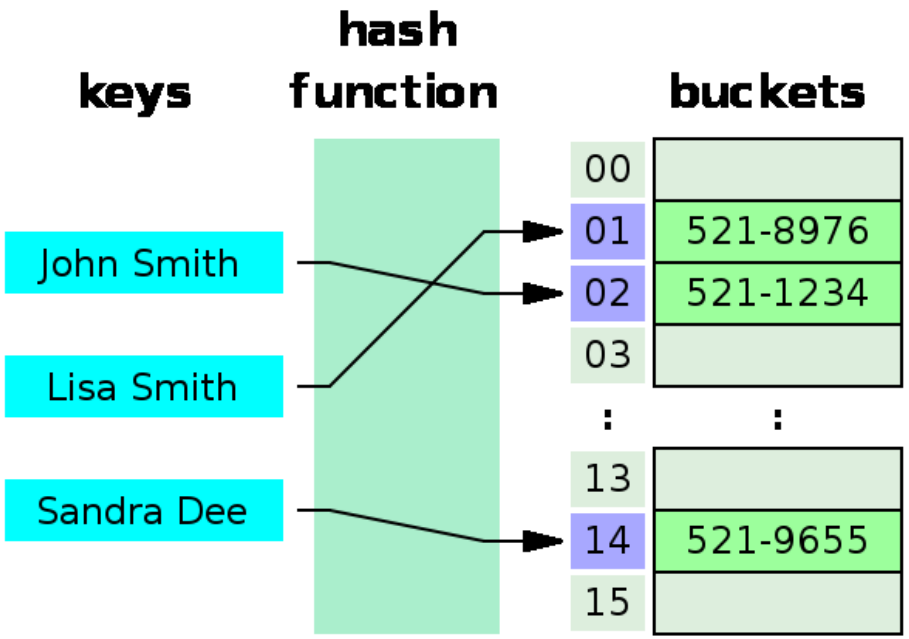
Il faut une fonction clé $\rightarrow$ adresse de "l'alvéole", où se trouve l'information ce sera la fonction de hachage

Fonction de hachage

Il faut une fonction clé $\rightarrow$ adresse de "l'alvéole", dans un bloc mémoire "suffisamment grand"

Fonction de hachage

Il faut une fonction clé $\rightarrow$ adresse de "l'alvéole", dans un bloc mémoire "suffisamment grand"



Exemple : stockage de numéros de téléphone

Fonctions de hachage

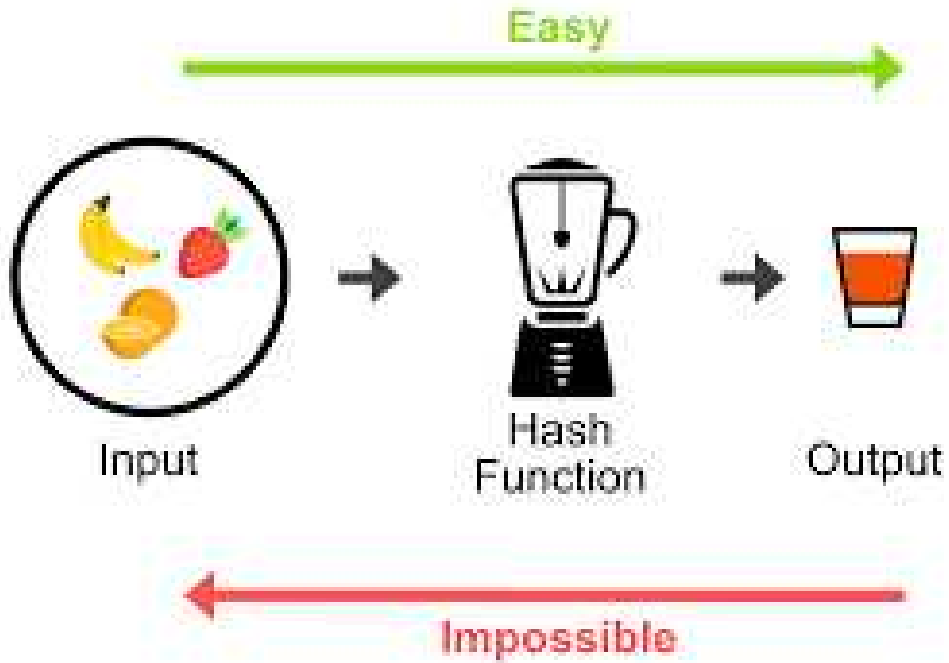
On exige qu'à partir d'une clef on trouve une adresse :

- facile à calculer,
- qui soit toujours la même (on a une véritable fonction)
- et qui extérieurement semble "aléatoire"

Fonctions de hachage

On exige qu'à partir d'une clef on trouve une adresse :

- facile à calculer,
- qui soit toujours la même (on a une véritable fonction)
- et qui extérieurement semble "aléatoire"



Fonctions de hachage

On exige qu'à partir d'une clef on trouve une adresse :

- facile à calculer,
- qui soit toujours la même (on a une véritable fonction)
- et qui extérieurement semble "aléatoire"

Caractéristiques de la fonction de hachage

Une fonction de hachage est déterministe, non prévisible, non réversible, avec peu de collisions.

Caractère non prévisible, non réversible :

clés ressemblantes $\rightarrow$ adresses sans trop de rapport
Ex D["mon ami Dominique"] et D["mon amie Dominique"]

Peu de collisions : idéalement une clé $\leftrightarrow$ une adresse mémoire (injectivité)

Problème : l'ensemble des "clés possibles" est énorme.
Conséquence : on va demander moins que l'injectivité

Plan

Fonction de hachage - exemples

Python propose une fonction de hachage concrète, à valeurs entières
- `hash(18)` donne 18
- `hash(10000000000)` donne 1410065412
- `hash('18')` donne 1275117679

Mais si on redémarre le PC les valeurs changent !! Cela signifie que le détail de la fonction de hachage est "tiré au sort" à chaque lancement d'un nouveau processus. Les entiers, en revanche, sont traités à part.

Fonction de hachage - exemples

Python propose une fonction de hachage concrète, à valeurs entières
- `hash(18)` donne 18
- `hash(10000000000)` donne 1410065412
- `hash('18')` donne 1275117679

Mais si on redémarre le PC les valeurs changent !! Cela signifie que le détail de la fonction de hachage est "tiré au sort" à chaque lancement d'un nouveau processus. Les entiers, en revanche, sont traités à part.

- Traitement spécifique pour les entiers : le modulo N

Je dispose de $N = 2^{31} - 1$ adresses mémoires, mais les clés peuvent prendre des valeurs entières très grandes. Alors on prend pour adresse "clé modulo N".
Dans ce cas, les collisions sont connues.

Fonction de hachage concrète "par tabulation"

Juste pour la culture !!!

Et avec des données plus complexes comme des chaînes de caractères ?

- couper la clef donnée en tronçons (8 bits)
- appliquer une transformation T aux tronçons
- obtenir le résultat en appliquant des XOR à ces résultats successifs

Fonction de hachage concrète "par tabulation"

Juste pour la culture !!!

Et avec des données plus complexes comme des chaînes de caractères ?

- couper la clef donnée en tronçons (8 bits)
- appliquer une transformation T aux tronçons
- obtenir le résultat en appliquant des XOR à ces résultats successifs

Reste à voir la transformation T : il y a en fait 8 transformations T_1 à T_8 et on applique alternativement chacune de ces 8 transformations à chaque tronçon.
Ces transformations sont "tirées au hasard" dans une phase "initialisation".
Concrètement on tire et on mémorise 256 valeurs $\leq N$ au hasard, 8 fois.

Ainsi on obtient toujours la même valeur de hachage une fois T fixé. Mais on obtient en fait des valeurs de hachage différentes si on exécute deux fois hash('chaine') dans différents processus.

Plan

Hachable, non hachable

Déf - Objets hachables

Un objet est "hachable" lorsqu'en utilisant la fonction de hachage à différents moments on retrouve toujours le même résultat.

- un objet simple (entier, flottant, booléen) est hachable

- un objet composé mutable est non hachable : liste, dictionnaire

- un objet composé mais non mutable est hachable : chaîne de caractères, tuple d'entiers

- sauf si certains des éléments sont mutables

OK car hachable	Non car pas hachable
D[18]	D[[18,19]]
D['dix-huit']	D[(3,[4,5])]
D[(18,19)]	

Plan

Collisions

Lorsqu'on remplit un dictionnaire progressivement on occupe les alvéoles

• La survenue de collisions est inévitable, même avec assez peu d'alvéoles remplies (c'est le problème des anniversaires)

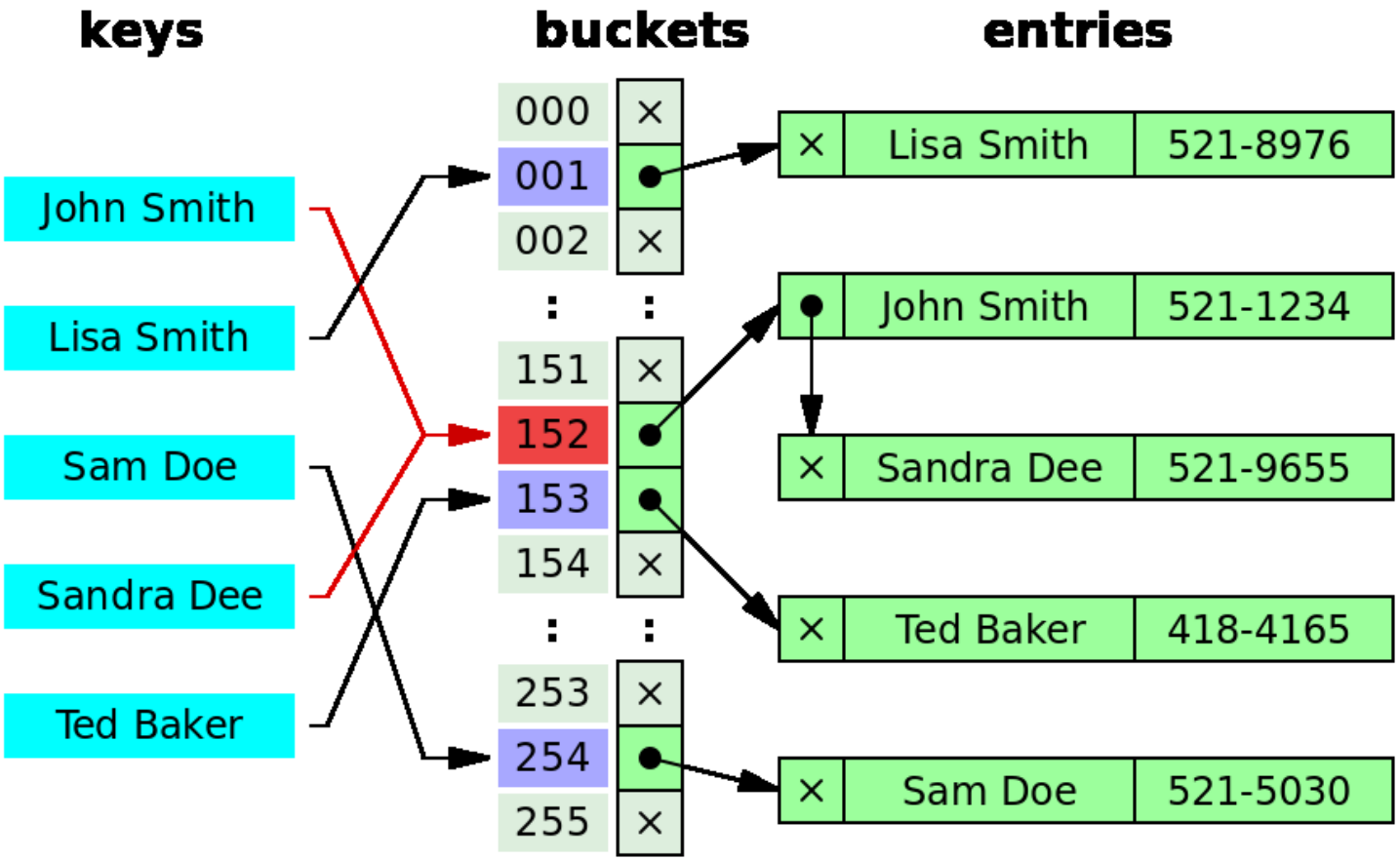
• Mais si la fonction de hachage est bien choisie, c'est aussi rare que possible : en $\frac{\text{nb\_clefs}}{N}$. Tout va bien tant que la taille N de la zone de stockage reste suffisamment grande par rapport au nombre de clés utilisées.

• Il reste la question

comment gérer les collisions

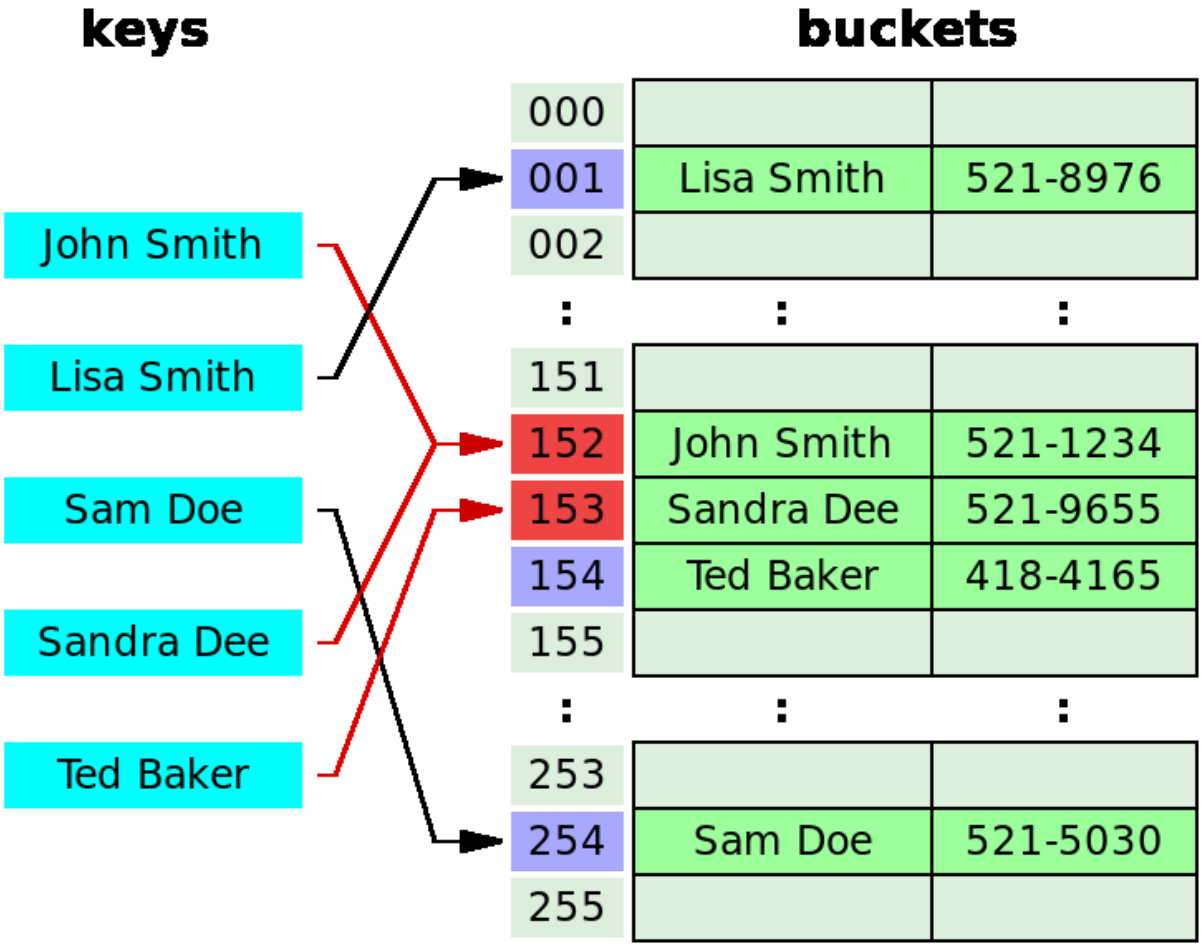
tout en conservant la complexité $O(1)$ pour les opérations élémentaires ?

Solution liste chaînée



Les différents enregistrements en collision sont listés ; on doit faire un parcours linéaire et trouver la bonne valeur de la clé.

Solution adressage ouvert



C'est grosso modo la même idée, mais sur place : on lit les enregistrements dans l'ordre jusqu'à trouver ou tomber sur une case "vide".

Note sur ces deux solutions

A cause des collisions, il faut répéter la clef

Quand il y a collision, il faut retrouver la bonne valeur; la clef doit être présente.
Les dictionnaires ont une complexité en O(1) pour lire/écrire/modifier/chercher par la clef. Mais à utilisation identique ils sont plus lourds que les tableaux.

On privilégiera donc les tableaux élémentaires quand les opérations de dictionnaire ne sont p)as nécessaires.

Plan

Rehachage

En cas de remplissage trop important (nombre de clefs utilisées proche de N), les collisions deviennent trop fréquentes, et trop compliquées à gérer. Il faut donc augmenter la zone globale de stockage (augmenter N), et le plus souvent, faire une recopie.
Comme pour les tableaux dynamiques, une recopie est coûteuse ($O(N)$), mais sur un grand nombre d'utilisations, la complexité amortie (moyennée) reste en $O(1)$.