Tables de hachage

Récapitulatif

Tableaux : accès efficace

Tableaux triés : accès efficace, recherche très efficace (recherche dichotomique)

Listes : insertion / délétion efficace

Question : peut-on avoir une structure de données qui soit performante pour toutes les opérations ?

Spoiler alert : oui, **les tables de hachage** qui sont un peu moins efficace que les tableaux pour l'accès mais beaucoup plus que les listes et un peu moins efficace que les listes pour l'insertion / délétion mais beaucoup plus que les tableaux.

Bonus : la structure est *très* efficace pour la recherche

Clef

Objectif : fournir un identifiant, appelé une **clef**, unique à une entité

Par exemple :

- Donner un numéro unique à chaque étudiant dans une université
- Donner un numéro unique à des livres dans une bibliothèque

But : rechercher rapidement si une entité existe en leur attribuant une **clef**

Problème : que faire s'il peut y avoir un grand nombre de valeurs pour la **clef** ?

Par exemple, à l'Université de Lorraine, la clef (le numéro d'étudiant) est de la forme : 32219666 (8 chiffres)

Solution non réaliste : utiliser un tableau de 100000000 indexé de 0 à 99999999

Fonction de hachage

Notations

- L'ensemble des valeurs possibles des clefs s'appelle **l'univers de clefs (U)**
- On n'utilise généralement qu'un sous-ensemble $K \subseteq U$ des clefs

Important : les clefs sont un choix du programmeur.

Question : que faire si on dispose d'un tableau de taille n pour stocker les éléments avec $n << |K| << |U|$?

Fonction de hachage : une fonction $h : U \to [0 \ldots n-1]$

Utilisation : déterminer l'indice où ranger l'élément liée une clef

Table de hachage

Un tableau de taille n où un objet x associé à une clef k est stocké à l'indice $h(k)$

les opérations suivantes sont définies pour les tables de hachage :

- **RECHERCHE :** pour une clef k retourner une référence sur un objet de la table de hachage ayant k comme clef ou indiquer qu'un tel objet n'existe pas
- **INSERTION :** étant donné un nouvel objet x, ajouter x à la table de hachage
- **SUPPRESSION :** pour une clef k, retirer un objet dont la clef est k de la table de hachage si un tel objet existe

Efficacité des opérations

- **Recherche** : efficace (quasi-constant) si la fonction de hachage et la taille de la table de hachage sont biens choisies et les données ne sont pas pathologiques
- **Insertion** : aussi efficace (constant) qu'un tableau
- **Suppression** : efficace (quasi-constant) si la fonction de hachage et la taille de la table de hachage sont biens choisies et les données ne sont pas pathologiques

Quand utiliser une table de hachage ?

Une table de hachage s'utilise si

- on a besoin d'une recherche rapide
- on travaille sur un ensemble d'objets qui change de manière dynamique
- on peut définir une clef unique pour chaque élément

Exemple d'application : déduplication

Situation : Un *grand nombre* de données arrivent une à la fois sous forme d'un flux, par exemple :

- lecture des données dans un large fichier stocké sur un disque ;
- parcourir le web et traiter des milliards de pages web ;
- suivre des paquets de données passant par un routeur de réseaux à grande vitesse ;
- suivez les visiteurs d'un site web.

But : le but est de garder qu'une seule occurence de chaque clef et de l'objet associé.

Exemple : dans le suivi de visiteur, l'adresse IP peut servir de clef et on ne garde qu'une fois chaque adresse IP.

Algorithme pour la déduplication avec une table de hachage

Quand un nouvel objet x associé à une clef k arrive :

1. Utiliser **RECHERCHE** pour vérifier si la table de hachage contient déjà un objet avec la clef k
2. Si ce n'est pas le cas, utiliser **INSERTION** pour ajouter x à la table de hachage

Exemple d'application : problème 2-SUM

Données

- un tableau A de n entiers
- un entier cible t

Objectif

Déterminer s'il existe deux nombres x, y dans A tel que $x + y = t$

ou de manière équivalente

Déterminer s'il existe deux indices i et j dans $[1, \ldots, n]$ tel que $A[i] + A[j] = t$

Algorithme brute-force

Essayer toutes les combinaisons possibles pour x et y

coût : de l'ordre de n^2 *itérations*

Question : peut-on faire mieux ?

Tentative 1

Observation : pour une valeur x fixée, y vaut forcément $t - x$

Principe : recherchez dans le tableau $t - x$

```
Pour i = 1 à n faire
  y := t - A[i]
  si A contient y alors // recherche séquentielle
    retourner "oui"
retourner "non"
```

coût : de l'ordre de n^2 *itérations*

Question : peut-on faire mieux ?

Tentative 2

Observation : on peut améliorer la recherche en utilisant un tableau trié

Principe : trier le tableau et utiliser une recherche dichotomique

```
trier A // par exemple par le tri fusion
Pour i = 1 à n faire
  y := t - A[i]
  si A contient y alors // recherche dichotomique
    retourner "oui"
retourner "non"
```

coût : de l'ordre de $n \log n$ *itérations*

Remarque : $n \log n$ est **significativement** mieux que n^2

Question : peut-on faire mieux ?

Tentative 3

Observation : améliorer la recherche

Principe : utilisation d'une table de hachage où la clef est l'entier

```
H := une table de hachage vide

Pour i = 1 à n faire
  INSERER A[i] dans H

Pour i = 1 à n faire
  y := t - A[i]
  si H contient y alors // utilisation de RECHERCHER
    retourner "oui"

retourner "non"
```

coût : de l'ordre de n itérations

Remarque : n est **significativement** mieux que $n \log n$ et un ordre de grandeur plus petit que n^2

Implémentation : idées de haut niveau

Les principales idées à envisager sont

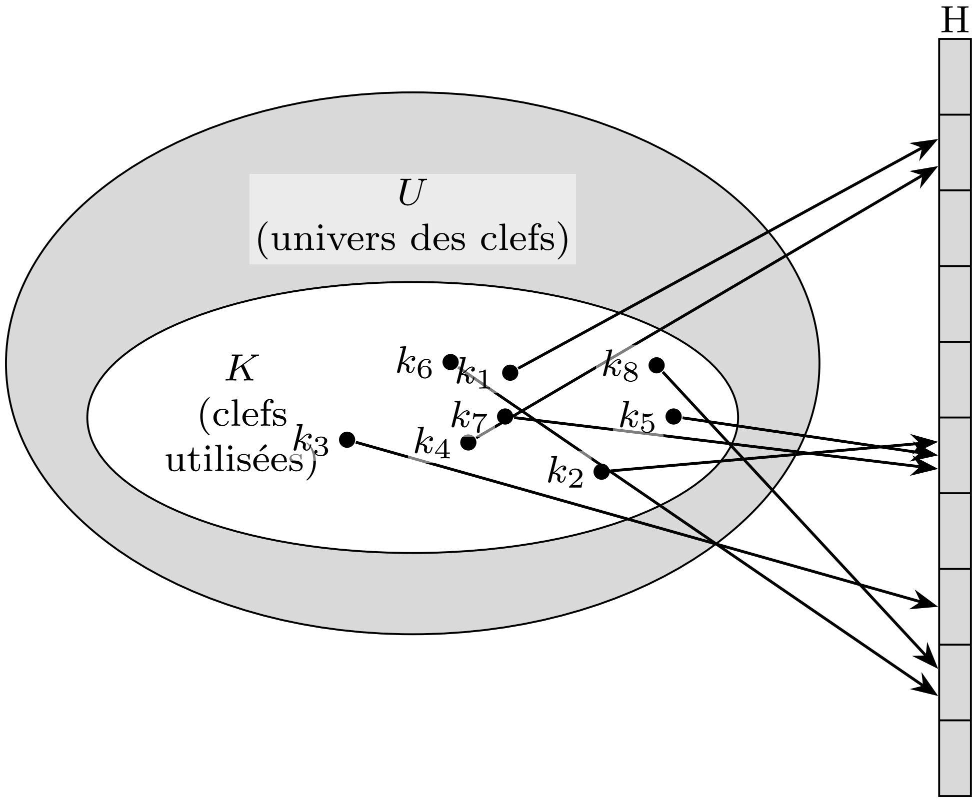
- les fonctions de hachage qui assigne à chaque clef un indice dans la table
- les **collisions** : cela arrive quand la fonction de hachage attribue le même indice à deux clefs différentes
- les stratégies de résolution des collisions : que faire quand une collision intervient

Remarque sur la taille de la table de hachage :

- le choix de la taille de la table est lié à la taille de U ou de K
- si U ou K sont de taille raisonnable, on peut utiliser un tableau de taille $|U|$ ou $|K|$ $\rightarrow$ le coût des opérations dépend alors de cette taille
- Si $|K|$ (et donc $|U|$) est trop grand, on peut utiliser une taille de table plus petite et ranger les objets dans des listes chaînées pour chaque indice $\rightarrow$ **RECHERCHE** et **SUPPRESSION** deviennent un peu moins efficace

Fonctions de hachage

une fonction de hachage $h : U \to \{0, 1, 2, \ldots, n-1\}$ affecte chaque clef de U à une position dans un tableau de taille n



Collision

Définition : deux clefs k_1 et k_2 de U sont en *collision* pour une fonction de hachage h si $h(k_1) = h(k_2)$.

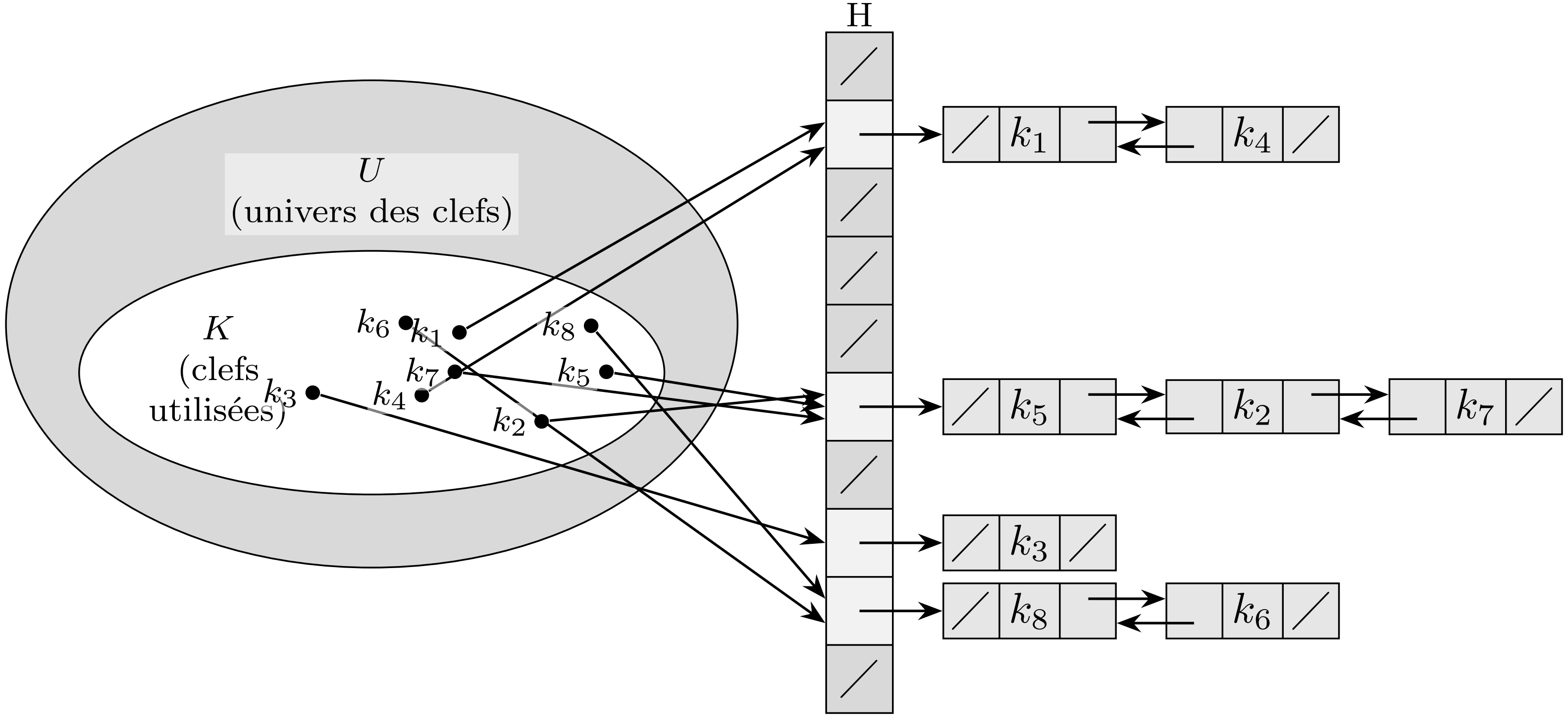
Principe des tiroirs de Dirichlet: si n chaussettes occupent m tiroirs et si $n > m$ alors au moins un tiroir doit contenir strictement plus d'une chaussette

Conséquence : les collisions sont inévitables ; il n'existe pas de fonction de hachage permettant d'éviter les collitsions si $|U| est strictement plus grand que la taille de la table de hachage

Question : que faire quand une collision se produit ?

Résolution des collisions : chaînage ou addressage fermé

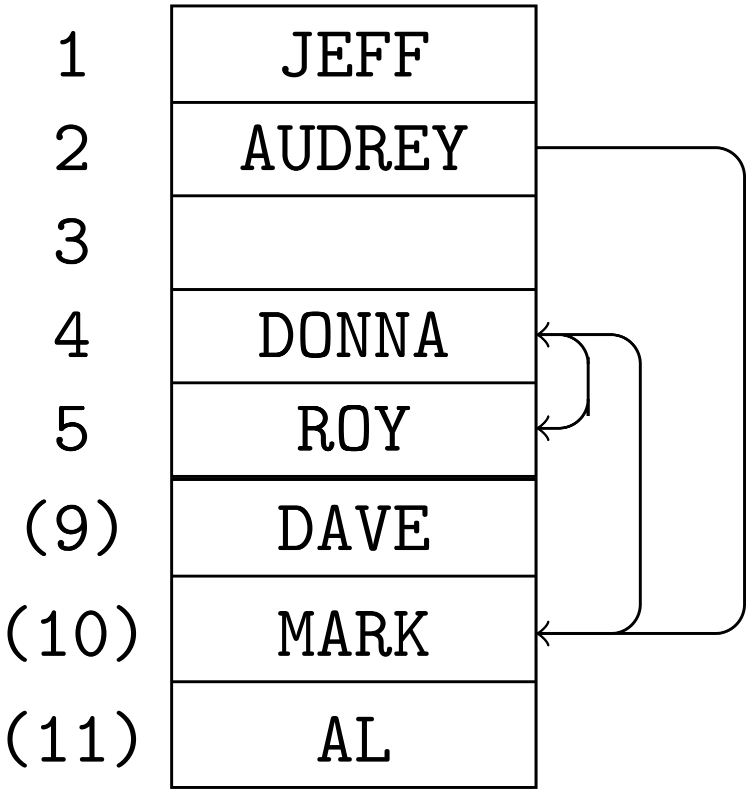
1. Garder une liste chaînée pour chaque position de la table de hachage
2. Pour RECHERCHER / INSERER / SUPPRIMER un objet avec une clef k, on effectue l'opération correspondante dans la liste chaînée se trouvant dans $H[h(k)]$



Résolution des collisions : addressage ouvert

Principe : Pour un objet associé à une clef k si la position $h(k)$ est déjà occupée, on recherche une place libre par **sondage**

Exemple : on souhaite ajouter **ROY** (qui est la clef et la valeur) ; $h(ROY) = 2$



Sondage

INSERTION : si la position est libre, on ajoute l'objet à cette position sinon on recherche une autre position par **sondage**

RECHERCHE / SUPPRESSION : il faut parcourir les positions dans le même ordre jusqu'à tomber sur l'objet recherché ou sur une position vide

Définition de la fonction de sondage : $h'(k, i)$ avec

- k : la clef
- i : le numéro de la tentative
- il faut que $h'(k, 0) = h(k)$
- h' doit permettre (si nécessaire de parcourir toutes les positions de la table)

Exemple: sondage linéaire $h'(k, i) = (h(k) + a \times i) \mod n$ avec $a \in \mathbb{N}^+$

Exemple: double hachage $h'(k, i) = (h(k) + i \times h_2(k)) \mod n$ avec h_2 une fonction de hachage

Conclusions

- le choix de la taille de la table de hachage a une influence sur l'efficacité pour éviter les collisions

$\Rightarrow$ **Compromis entre l'espace mémoire et le nombre de collisions**

- le choix de la fonction de hachage a une très forte influence sur l'efficacité pour éviter les collisions
- le choix du sondage dans un adressage ouverta une influence sur l'efficacité

$\Rightarrow$ **la définition de ces fonctions est difficile et la structure est présente dans la plupart des librairies des langages modernes**