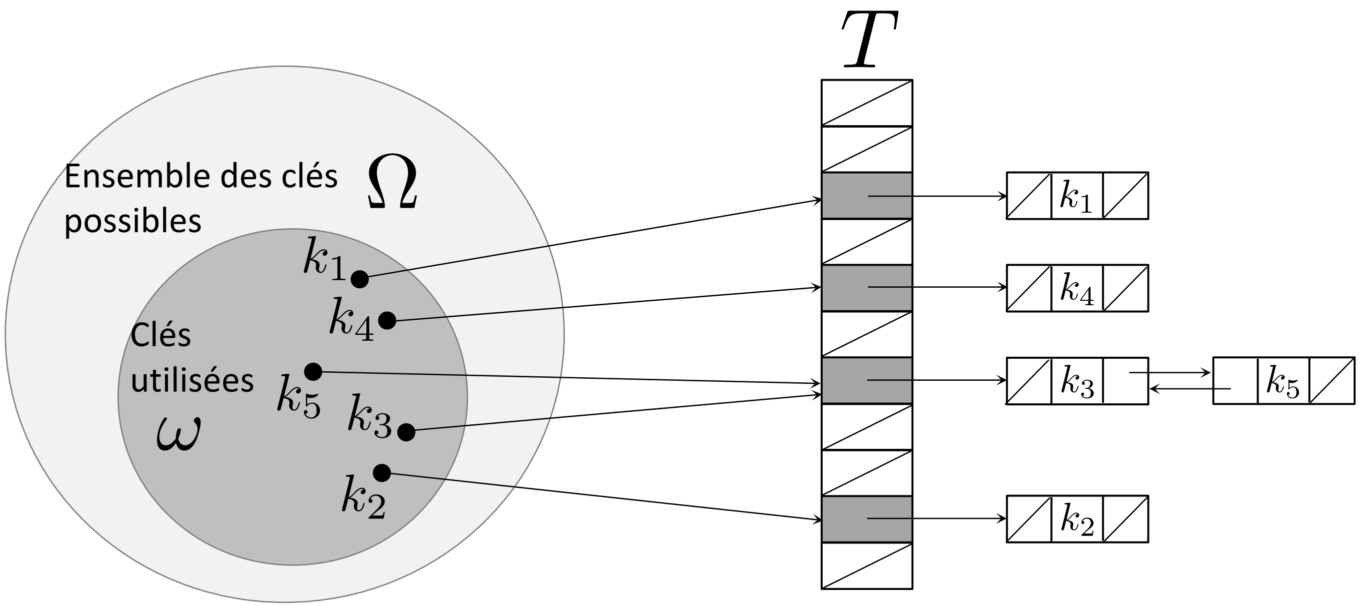
Algorithmique

Table de hachage



INSTITUT NATIONAL DES SCIENCES APPLIQUÉES LYON

Eric Guérin
INSA de Lyon – Département Informatique
3IF

Introduction

- Pour l'instant
  - Dictionnaire au mieux log(n)
  - Peut-on faire mieux ?
- Changement complet de philosophie
- Clé -> fonction(clé) = indice dans un tableau
- Fonction de hachage évaluée en temps constant

Tableau à accès direct

- Clé du dictionnaire = indice dans le tableau
  - Cela suppose que la clé est un entier
- Si la clé est utilisée, on stocke un pointeur dans la case correspondante
- Sinon, pointeur nul dans la case

Tableau à accès direct

- **Avantage** : rapide (temps constant)
- **Inconvénient** : la taille du tableau est liée à l'amplitude des données
- Or bien souvent **le cardinal de l'ensemble des clés utilisées est bien inférieur à l'amplitude des données**
- Exemple : la clé est une année de naissance

Table de hachage

- Une fonction est intercalée entre la clé et l'indice du tableau
- La taille du tableau est bien inférieure à celle de l'ensemble des valeurs de clés possibles

$$h : \Omega \to \{0, \ldots, m-1\}$$

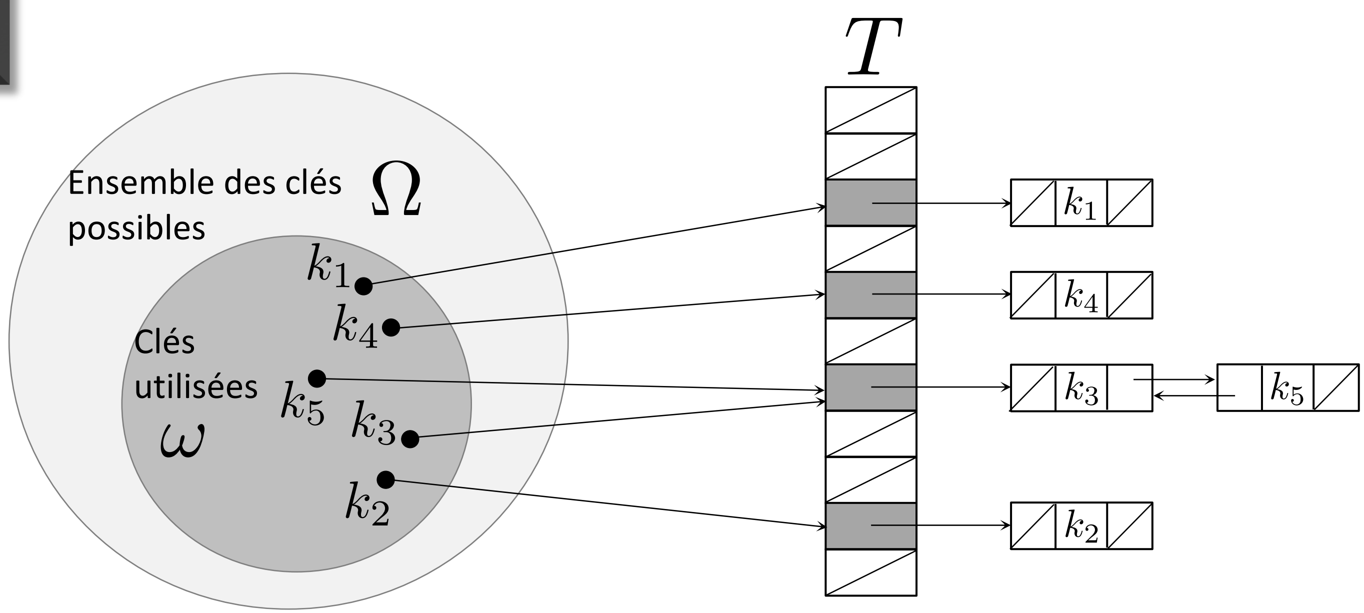
- La case utilisée est $T[h(k)]$ au lieu de $T[k]$
- **Contrainte** : construire une fonction dont le résultat est un entier entre 0 et m-1

Fonction de hachage

- Elle ne peut pas être injective
  - Une valeur de l'ensemble cible peut avoir plusieurs antécédents
- Donc deux clés peuvent donner la même valeur de hachage
  - Collision !
- Deux manières de gérer la collision
  - Adressage fermé (chaque case contient plusieurs données)
  - Adressage ouvert (on trouve une autre case automatiquement)

Adressage fermé - chaînage

- Chaque case du tableau contient plusieurs informations
  - Liste chaînée par exemple



Adressage fermé - chaînage

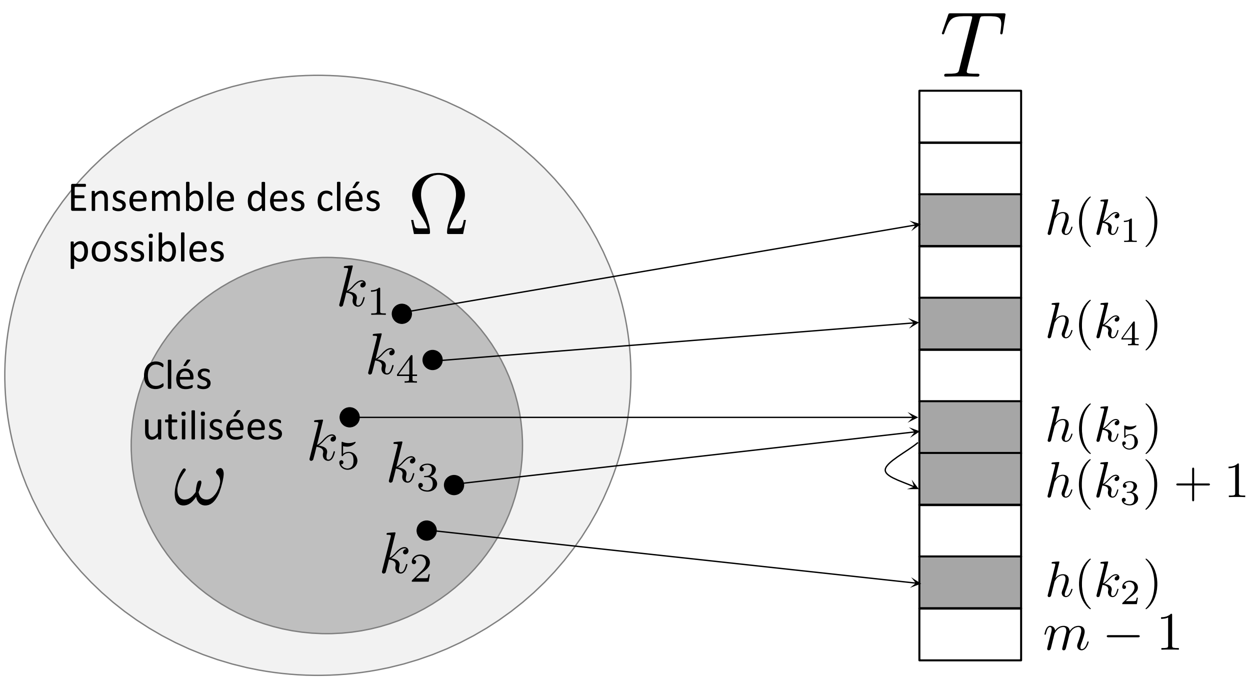
- Insertion
  - Temps constant pour accéder à la liste chaînée
  - Temps linéaire pour accéder à la bonne case de la liste
- Complexité dépend du rapport entre :
  - u le nombre de clés effectivement utilisées
  - m le nombre de cases du tableau
- Si u = O(m) alors le rapport est de l'ordre de 1 donc optimal garanti
- Correspond au cas où le nombre de cases du tableau est choisi en fonction de la taille des données à représenter

Adressage fermé - chaînage

- Rien n’oblige à utiliser une liste chaînée derrière la table de hachage
- Pourquoi pas un arbre binaire de recherche ?
- Exercice : discuter de l’avantage de l’ABR par rapport à la liste chaînée

Adressage ouvert

- Plus complexe
  - Consiste à trouver une autre case de libre dans le tableau



Adressage ouvert

- Stratégie
  - Prendre la première case vide
  - Simple mais efficace
  - D'une manière générale, fonction de hachage qui prend deux arguments : (clé,rang)
  - Si T[h(k,0)] est occupée, on essaye T[h(k,1)] et ainsi de suite

Adressage ouvert

- Et la suppression ?
  - **Problème** : une case devient libre et vient masquer les collisions suivantes
  - **Solution** : utiliser une valeur particulière qui indique que la case a été supprimée
- En pratique, lorsqu'on supprime on garde les mêmes performances de recherches alors que le nombre d'éléments a diminué

Fonction de hachage

- **Bonne propriété 1** : distribution uniforme des indices du tableau donc des valeurs de la fonction
- **Bonne propriété 2** : rapide à évaluer !
- Au préalable : il faut coder sa clé, la transformer en une valeur entière naturelle
- Exemple, une chaîne de caractères

$$\mathrm{code}(\mathrm{chaine}) = \sum_{i=0}^{\mathrm{taille}(\mathrm{chaine})} \mathrm{chaine}[i]\ 128^{i}$$

Fonction de hachage

$$\mathrm{code}(\mathrm{chaine}) = \sum_{i=0}^{\mathrm{taille}(\mathrm{chaine})} \mathrm{chaine}[i]\ 128^{i}$$

Note d'implémentation

Faire une rotation de bits plutôt qu'un décalage simple :

```
unsigned int shift_rotate(unsigned int val, unsigned int n)
{
  n = n%(sizeof(unsigned int)*8);
  return (val<<n) | (val>> (sizeof(unsigned int)*8-n));
}
```

Fonction de hachage

- Méthode basée sur le reste de la division

$$h(k) = k \bmod m$$

- Inconvénient
  - N'utilise que les bits de poids faible de k
  - Exemple : s'il s'agit d'une chaîne de caractères et que m = 128 alors h ne dépend que de chaine[0]
  - Toutes les chaînes qui commencent par la même lettre seront stockées dans le même emplacement

- Solution : ne pas prendre une puissance de 2

Fonction de hachage

- Méthode basée sur les parties entières

$$h(k) = \lfloor m \,\mathrm{frac}(\lambda\, k) \rfloor$$

- Avec $\lambda \in [0, 1]$ et $\mathrm{frac}(x) = x - \lfloor x \rfloor$
- Choix de λ important, de préférence un nombre irrationnel

Fonction de hachage

- Exercice : pour $\lambda = \frac{1}{2}$ expliciter les valeurs que peuvent prendre la fonction
  - Est-ce un bon choix ?

- En pratique, une valeur qui marche bien

$$\lambda = \frac{\sqrt{5} - 1}{2}$$

Comparaison

- Adressage fermé
  - Avantages
    - Nombre quelconque d'éléments et de collisions
    - Performances stables car la suppression n'engendre pas de détérioration
  - Inconvénient
    - Surcoût mémoire de gestion de la liste

Comparaison

- Adressage ouvert
  - Avantages
    - Pas de surcoût mémoire
    - Rapide
  - Inconvénient
    - Choix de la fonction de hachage délicat (pour éviter les valeurs proches)
    - Taille limitée et non extensible
    - Suppression dégrade les performances de recherche

Conclusion

- Il faut tenir compte des contraintes
- Si vous utilisez une table de hachage regardez comment elle est construite de manière interne
  - Paramétrage ?
- Choix avec d’autres implémentations de dictionnaires
  - Table de hachage : pas d’ordre
  - A utiliser lorsque les données n’ont pas besoin d’être triées