STR3 : tables de hachage

loig.jezequel@univ-nantes.fr

Dictionnaires

Tableaux

Données indexées par des entiers consécutifs : $t[i]$ est la donnée associée à l'entier i.

- i est un indice,
- les indices sont des entiers (consécutifs et partant de 0).

Dictionnaires

Données indexées par des données de type quelconque : par exemple $t[\text{"}bonjour\text{"}]$ est la donnée associée à la chaîne "*bonjour*".

- "*bonjour*" est une clé,
- toutes les clés d'un dictionnaire ont le même type.

Dictionnaires en Go : map

Déclaration

Exemple : var m map[string]bool

- string est le type des clés, on peut utiliser n'importe quel type pour les clés du moment qu'il est comparables,
- bool est le type des données, on peut utiliser n'importe quel type pour les données.

Initialisation

Exemple : m = make(map[string]bool)
Exemple : m = map[string]bool{}

- Comme pour les tableaux, deux initialisations possibles (avec ou sans l'utilisation de make)

Code Go : mapcreation.go

Dictionnaires en Go : map (suite)

Ajouter des éléments à la création

Exemple : m = map[string]bool{"bonjour": true, "salut": false}

- on indique des couples clé: élément,
- séparés par des virgules.

Ajouter des éléments plus tard

Exemple : m["yo"] = false

- un nouveau couple (clé, élément) est ajouté dans la map,
- si la clé existe déjà, l'élément associé est remplacé.

Code Go : mapaddelement.go

Dictionnaires en Go : map (suite, suite)

Accéder à une donnée à partir d'une clé

Exemple : m["bonjour"]

- si la clé est associée à une valeur (existe), on obtient la valeur,
- si la clé n'existe pas, on obtient la valeur par défaut associée au type des valeurs.

Vérifier l'existence d'une clé

Exemple : b, exists = m["bonjour"]

- l'accès à une clé retourne en fait deux valeurs : la valeur de l'élément associée (ou une valeur par défaut si celui-ci n'existe pas) et un booléen qui indique si la clé existe.

Code Go : mapgetelement.go

Dictionnaires en Go : map (suite, suite, suite)

Supprimer un élément

Exemple : delete(m, "bonjour")

- delete prend en paramètres une map et une clé et supprime la clé de la map.

Code Go : mapdeleteelement.go

Parcourir les clés et les valeurs

- Mot clé range, déjà vu sur les tableaux,
- permet de parcourir tous les couples (clé, valeur) un par un,
- on ne sait rien sur l'ordre dans lequel ces couples sont parcourus (peut changer d'une fois sur l'autre).

Code Go : maprange.go

Intérêt des dictionnaires

Économie de mémoire

Si on a un ensemble de données indicées, mais avec beaucoup de trous entre les indices, la représentation par un tableau occupe beaucoup d'espace inutilement.

- Un dictionnaire bien implanté peut occuper beaucoup moins de mémoire en ne contenant que les données utiles.

Économie de temps

Un dictionnaire bien implanté permet de vérifier la présence d'une clé (et donc d'accéder à la valeur correspondante) en temps constant en moyenne : le temps d'accès ne dépend pas ou très peu du nombre de clés ou de la taille de l'espace des clés.

- Test efficace d'appartenance d'une donnée à un ensemble.

Implantation des dictionnaires : adressage direct

Présupposé

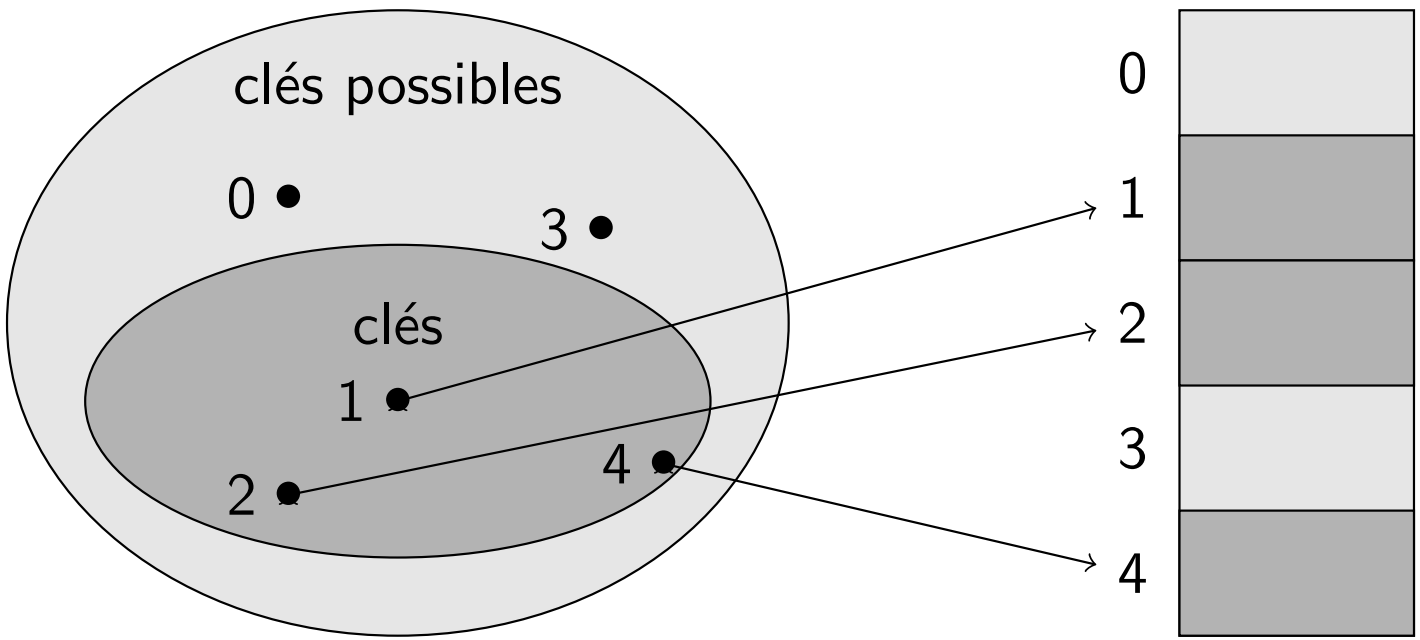
L'espace des clés est petit.

Opérations (immédiat)

Recherche, insertion, suppression

Principe

On numérote les clés par les indices d'un tableau dont le nombre de case correspond au nombre de clés possibles.



Implantation des dictionnaires : tables de hachage

Utilisation

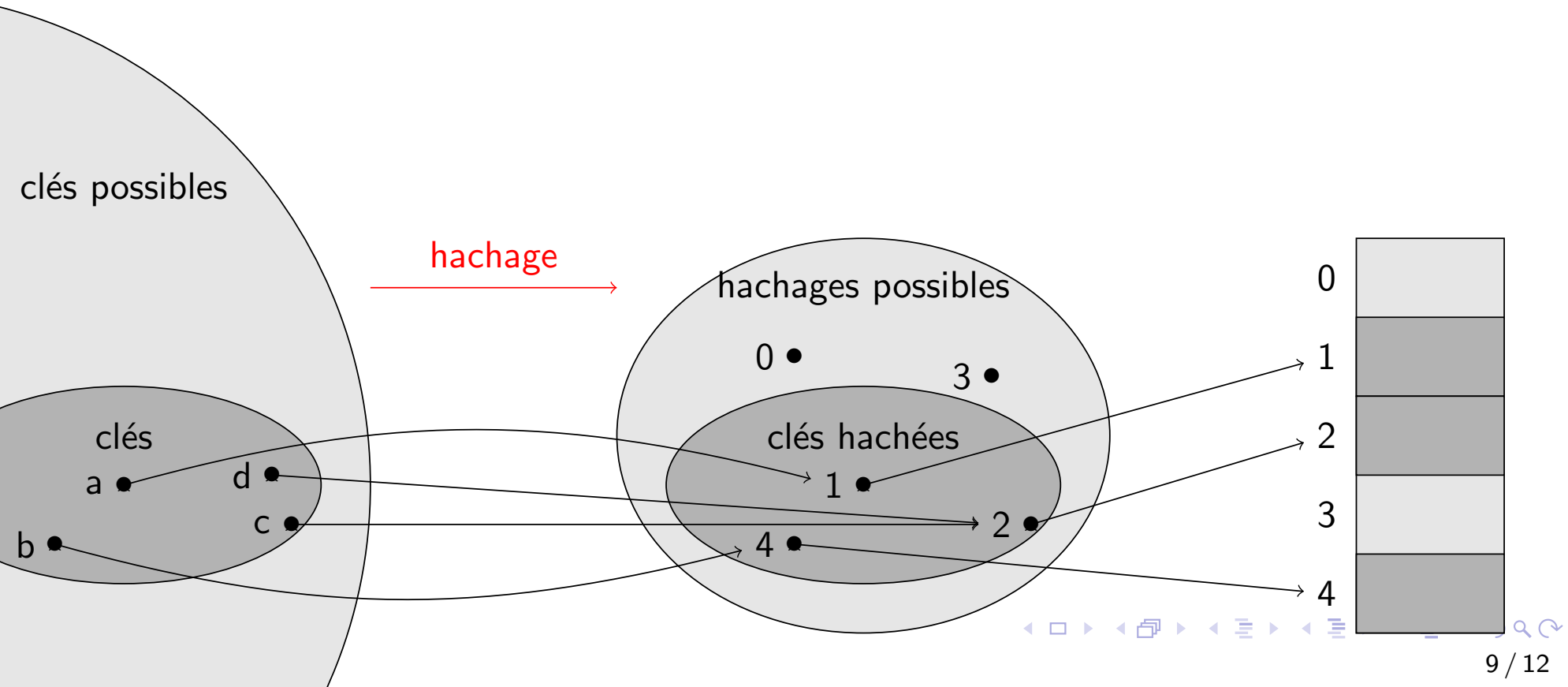
Quand l'espace des clés est grand et qu'on en utilise peu.

Opérations (collisions ?)

Recherche, insertion, suppression

Principe

On utilise une fonction de hachage pour réduire le nombre de clés possibles, puis adressage direct à partir des clés hachées.



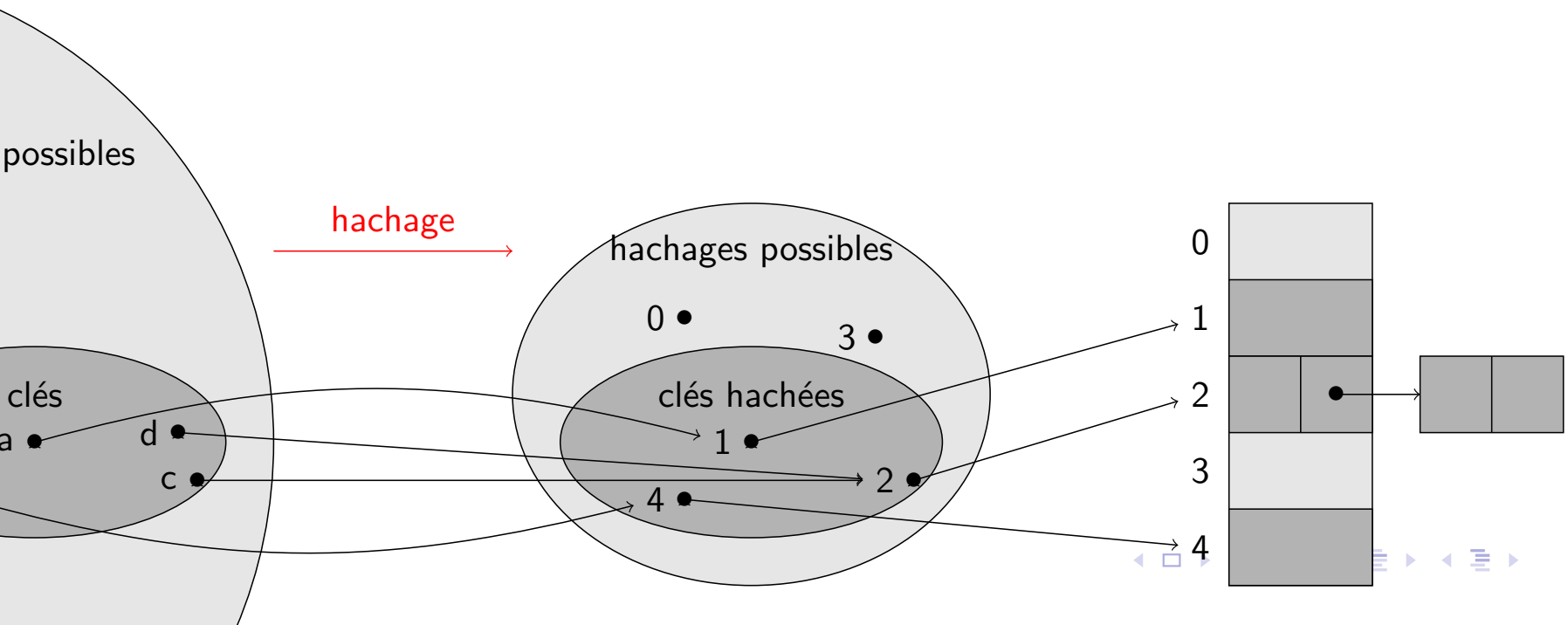
Tables de hachage, résoudre le problème des collisions

Il y a forcément des collisions

Principe des tiroirs et des chaussettes : en projetant un grand ensemble sur un petit ensemble on crée obligatoirement des collisions.

Solution classique

Les valeurs dont la clé a le même hachage sont stockées dans une liste chaînée.



Fonctions de hachage : propriétés

Propriétés attendues

Objectif : minimiser les risques de collisions

- hachage uniforme : pour tout hachage k, la probabilité qu'une clé prise au hasard ait pour hachage k doit être le plus proche possible de $1/m$ où m est le nombre de hachages différents (la taille de l'espace de hachage),
- hachage rapide : hacher une clé demande peut d'opérations, on devra le faire très souvent.

Clés qui ne sont pas des entiers naturels

En général, les fonctions de hachage hachent les entiers naturels. Quand les clés ne sont pas des entiers, il faut trouver un moyen de les interpréter comme des entiers naturels (ce qui est en général assez simple, car toute donnée est codée en binaire, donc sous forme d'entiers).

Fonctions de hachage : exemple

Méthode de la division

Pour une table de hachage de taille m on peut calculer le hachage d'une clé k quelconque par la formule suivante :

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

Efficace ?

- Hacher une clé est très rapide (une seule division),
- on peut en général choisir un m qui rend le hachage uniforme (ou presque) en fonction de nos clés,
- prendre pour m un nombre premier, le plus éloignés possible d'une puissance de 2 est souvent un bon choix si on ne sait rien de la répartition de nos clés.