Dictionnaires

Lycée Thiers

Crédits

- Tableaux associatifs (Wikipedia)

Crédits

- Tableaux associatifs (Wikipedia)
- openclassrooms

Crédits

- Tableaux associatifs (Wikipedia)
- openclassrooms
- Informatique Tronc Commun (Serge Bays - Ellipse)

Tableaux associatifs

Définition

Un *tableau associatif* (aussi appelé *dictionnaire* ou *table d'association*) est un type de données associant à un ensemble de clefs un ensemble correspondant de valeurs.
Chaque clef est associée à une valeur : un tableau associatif correspond donc à une application en mathématiques.

- Grossièrement, le tableau associatif est une généralisation du tableau : alors que le tableau traditionnel associe des entiers consécutifs à des valeurs d'un certain type, le tableau associatif associe des valeurs d'un type arbitraire à des valeurs d'un autre type.

Tableaux associatifs

Exemples de la vie courante :

- dictionnaire franco-anglais. Les clés sont les mots en français et les valeurs les traductions en anglais.

Tableaux associatifs

Exemples de la vie courante :

- dictionnaire franco-anglais. Les clés sont les mots en français et les valeurs les traductions en anglais.
- Annuaire : les clefs sont les noms des usagers, les valeurs sont les numéros de téléphone.

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;
  - modification : association d'une nouvelle valeur à une ancienne clef ;

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;
  - modification : association d'une nouvelle valeur à une ancienne clef ;
  - suppression : suppression d'une clef et de la valeur correspondante ;

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;
  - modification : association d'une nouvelle valeur à une ancienne clef ;
  - suppression : suppression d'une clef et de la valeur correspondante ;
  - recherche : détermination de la valeur associée à une clef, si elle existe ;

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;
  - modification : association d'une nouvelle valeur à une ancienne clef ;
  - suppression : suppression d'une clef et de la valeur correspondante ;
  - recherche : détermination de la valeur associée à une clef, si elle existe ;
- On souhaite que ces opérations aient un coup amorti en $O(1)$ et une complexité au pire (rarement atteinte) en $O(n)$ (où n est le nombre de clés)

Opérations courantes

- Opérations dont la complexité temporelle attendue est en $O(1)$:
  - ajout : association d'une nouvelle valeur à une nouvelle clef ;
  - modification : association d'une nouvelle valeur à une ancienne clef ;
  - suppression : suppression d'une clef et de la valeur correspondante ;
  - recherche : détermination de la valeur associée à une clef, si elle existe ;
- On souhaite que ces opérations aient un coup amorti en $O(1)$ et une complexité au pire (rarement atteinte) en $O(n)$ (où n est le nombre de clés)
- Souvent, on s'accorde la possibilité de boucler sur les clés du dictionnaire.

Ordre des clés

- Dans un tableau (considéré comme un dictionnaire dont les clés sont des entiers), il y a un ordre naturel pour les clefs : celui des entiers.

Ordre des clés

- Dans un tableau (considéré comme un dictionnaire dont les clés sont des entiers), il y a un ordre naturel pour les clefs : celui des entiers.
- Mais la notion théorique de dictionnaire n'impose en rien que l'ensemble des clés possibles soit ordonné.

Ordre des clés

- Dans un tableau (considéré comme un dictionnaire dont les clés sont des entiers), il y a un ordre naturel pour les clefs : celui des entiers.
- Mais la notion théorique de dictionnaire n'impose en rien que l'ensemble des clés possibles soit ordonné.
- Il faut bien comprendre que même si l'ensemble des clés est ordonné, le parcours des clés ne respecte pas en général ce caractère.

Structures courantes

- L'implémentation la plus simple est la *liste d'association* (liste de couples clé, valeur)

```
1 let l = [("toto",45.);("titi",89.); ("XX",203.5)]
```

Insertion en $O(1)$, recherche en $O(n)$ au pire et en moyenne.

Structures courantes

- L'implémentation la plus simple est la *liste d'association* (liste de couples clé, valeur)

```
1 let l = [("toto",45.);("titi",89.); ("XX",203.5)]
```

  Insertion en $O(1)$, recherche en $O(n)$ au pire et en moyenne.

- Les dictionnaires sont le plus souvent utilisés lorsque l'opération de recherche est la plus fréquente. Pour cette raison, la conception privilégie le plus souvent cette opération, au détriment de l'efficacité de l'ajout et de l'occupation mémoire par rapport à d'autres structures de données.

Structures courantes

- L'implémentation la plus simple est la *liste d'association* (liste de couples clé, valeur)

```
1 let l = [("toto",45.);("titi",89.); ("XX",203.5)]
```

  Insertion en $O(1)$, recherche en $O(n)$ au pire et en moyenne.

- Les dictionnaires sont le plus souvent utilisés lorsque l'opération de recherche est la plus fréquente. Pour cette raison, la conception privilégie le plus souvent cette opération, au détriment de l'efficacité de l'ajout et de l'occupation mémoire par rapport à d'autres structures de données.
- Deux structures sont particulièrement efficaces pour représenter les tableaux associatifs : la *table de hachage* et l'*arbre équilibré*.

Implémentation

Comparaison des complexités temporelles

Opérations	Tables de hachage	Arbre équilibré
Insertion	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.
recherche	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.

- En général, les tables de hachage ont une représentation plus compacte en mémoire.

Implémentation

Comparaison des complexités temporelles

Opérations	Tables de hachage	Arbre équilibré
Insertion	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.
recherche	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.

- En général, les tables de hachage ont une représentation plus compacte en mémoire.
- Les tables de hachage imposent la création (souvent difficile) d'une *fonction de hachage*, les arbres équilibrés ont juste besoin d'un ordre total sur les clefs.

Implémentation

Comparaison des complexités temporelles

Opérations	Tables de hachage	Arbre équilibré
Insertion	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.
recherche	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.

- En général, les tables de hachage ont une représentation plus compacte en mémoire.
- Les tables de hachage imposent la création (souvent difficile) d'une *fonction de hachage*, les arbres équilibrés ont juste besoin d'un ordre total sur les clefs.
- L'ensemble des clés n'a pas besoin d'un ordre total pour les tables de hachage.

Implémentation

Comparaison des complexités temporelles

Opérations	Tables de hachage	Arbre équilibré
Insertion	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.
recherche	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.

- En général, les tables de hachage ont une représentation plus compacte en mémoire.
- Les tables de hachage imposent la création (souvent difficile) d'une *fonction de hachage*, les arbres équilibrés ont juste besoin d'un ordre total sur les clefs.
- L'ensemble des clés n'a pas besoin d'un ordre total pour les tables de hachage.
- Les arbres équilibrés s'implémentent bien avec des structures de données persistantes (dans une optique de programmation fonctionnelle) pas les tables de hachage).

Implémentation

Comparaison des complexités temporelles

Opérations	Tables de hachage	Arbre équilibré
Insertion	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.
recherche	$O(1)$ moyenne, $O(n)$ pire	$O(\log n)$ moyenne et pire.

- En général, les tables de hachage ont une représentation plus compacte en mémoire.
- Les tables de hachage imposent la création (souvent difficile) d'une *fonction de hachage*, les arbres équilibrés ont juste besoin d'un ordre total sur les clefs.
- L'ensemble des clés n'a pas besoin d'un ordre total pour les tables de hachage.
- Les arbres équilibrés s'implémentent bien avec des structures de données persistantes (dans une optique de programmation fonctionnelle) pas les tables de hachage).
- En OCaml, les 2 existent.

Objectif et fonctionnement

- Une *fonction de hachage* répartit les paires clé–valeur dans un tableau d'*alvéoles*. Une case du tableau est une alvéole : parfois c'est juste une association, d'autres fois une liste d'associations.

Objectif et fonctionnement

- Une *fonction de hachage* répartit les paires clé–valeur dans un tableau d'*alvéoles*. Une case du tableau est une alvéole : parfois c'est juste une association, d'autres fois une liste d'associations.
- Elle transforme une clé en une valeur de hachage, ce qui donne la position dans un tableau.

Objectif et fonctionnement

- Une *fonction de hachage* répartit les paires clé–valeur dans un tableau d'*alvéoles*. Une case du tableau est une alvéole : parfois c'est juste une association, d'autres fois une liste d'associations.
- Elle transforme une clé en une valeur de hachage, ce qui donne la position dans un tableau.
- Son calcul se fait parfois en deux temps :

Objectif et fonctionnement

- Une *fonction de hachage* répartit les paires clé–valeur dans un tableau d'*alvéoles*. Une case du tableau est une alvéole : parfois c'est juste une association, d'autres fois une liste d'associations.
- Elle transforme une clé en une valeur de hachage, ce qui donne la position dans un tableau.
- Son calcul se fait parfois en deux temps :
  - Une fonction de hachage particulière à l'application est utilisée pour produire un nombre entier à partir de la clé ;

Objectif et fonctionnement

- Une *fonction de hachage* répartit les paires clé–valeur dans un tableau d'*alvéoles*. Une case du tableau est une alvéole : parfois c'est juste une association, d'autres fois une liste d'associations.
- Elle transforme une clé en une valeur de hachage, ce qui donne la position dans un tableau.
- Son calcul se fait parfois en deux temps :
  - Une fonction de hachage particulière à l'application est utilisée pour produire un nombre entier à partir de la clé ;
  - Ce nombre entier est converti en une position possible de la table, en général en calculant le reste modulo la taille de la table.

Objectif et fonctionnement

Si la clé n'est pas un entier naturel, on fait en sorte de la considérer comme telle. Par exemple, pour les chaînes de caractères :

- on considère chaque caractère comme un nombre (par exemple avec le code ASCII) ;

Objectif et fonctionnement

Si la clé n'est pas un entier naturel, on fait en sorte de la considérer comme telle. Par exemple, pour les chaînes de caractères :

- on considère chaque caractère comme un nombre (par exemple avec le code ASCII) ;
- puis on le combine tous les nombres obtenus par une fonction rapide (souvent le XOR -OU EXCLUSIF-).
  Les divisions sont à éviter en raison de leur relative lenteur sur certaines machines.

Présentation simplifiée

- On dispose donc d'une fonction *h* dite de *hachage* qui va de l'ensemble des clés vers l'ensemble des indexes d'un tableau de couples (clés,valeurs) (ces derniers sont appelés *alvéoles*).

Présentation simplifiée

- On dispose donc d'une fonction h dite de *hachage* qui va de l'ensemble des clés vers l'ensemble des indexes d'un tableau de couples (clés,valeurs) (ces derniers sont appelés *alvéoles*).
- Lorsqu'on ajoute une nouvelle association (John Smith,+1555-8976), la valeur h(John Smith) est calculée. C'est l'indice d'une case du tableau d'association.

Présentation simplifiée

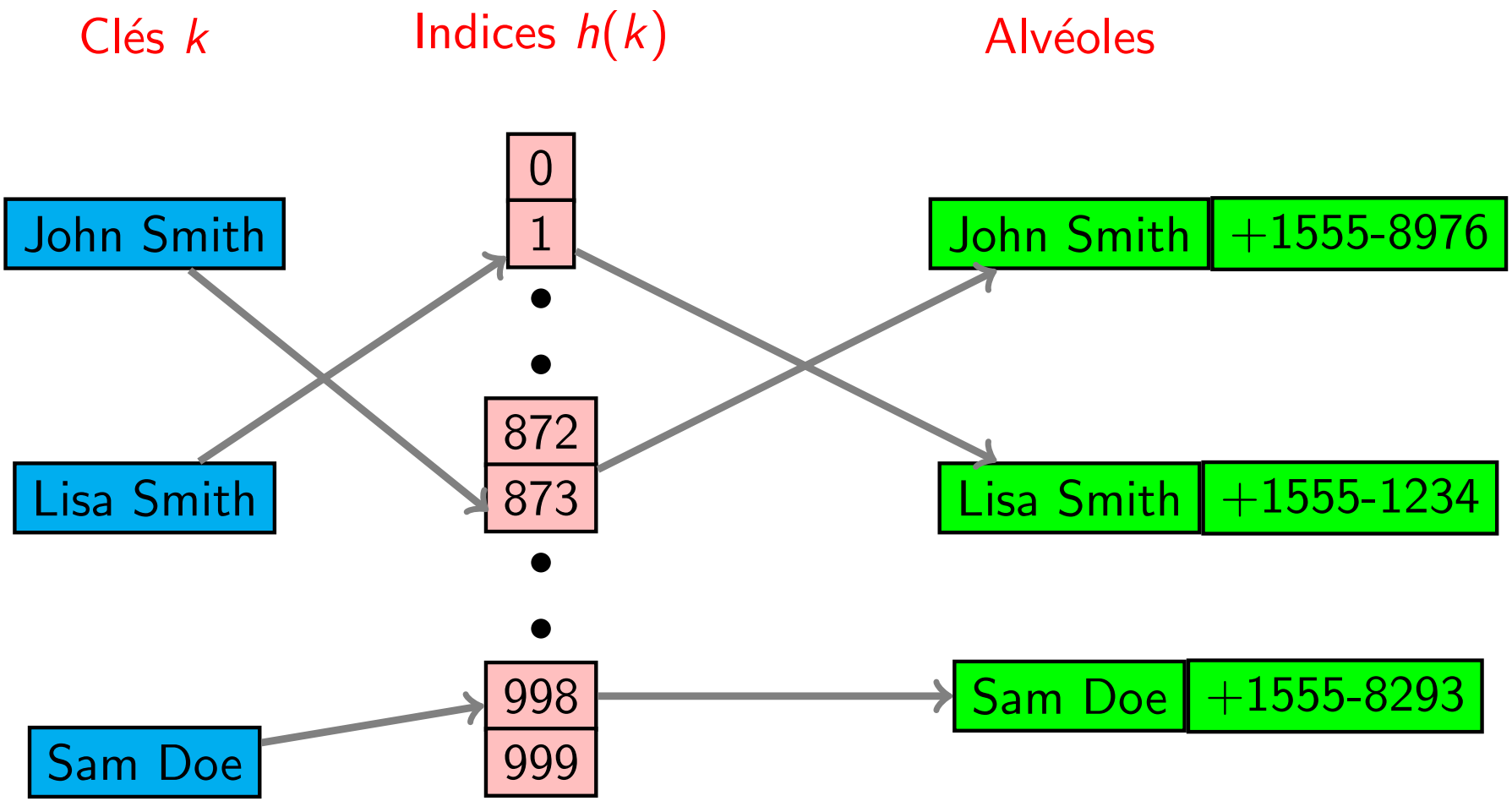
- On dispose donc d'une fonction h dite de *hachage* qui va de l'ensemble des clés vers l'ensemble des indexes d'un tableau de couples (clés,valeurs) (ces derniers sont appelés *alvéoles*).
- Lorsqu'on ajoute une nouvelle association (John Smith,+1555-8976), la valeur h(John Smith) est calculée. C'est l'indice d'une case du tableau d'association.
- A la case h(John Smith), on met le tuple (John Smith,+1555-8976).

Présentation simplifiée

- On dispose donc d'une fonction h dite de *hachage* qui va de l'ensemble des clés vers l'ensemble des indexes d'un tableau de couples (clés,valeurs) (ces derniers sont appelés *alvéoles*).
- Lorsqu'on ajoute une nouvelle association (John Smith,+1555-8976), la valeur h(John Smith) est calculée. C'est l'indice d'une case du tableau d'association.
- A la case h(John Smith), on met le tuple (John Smith,+1555-8976).
- Si la fonction de hachage est injective (ce qui n'arrive presque jamais), deux clés différentes ont deux valeurs de hachage différentes et on n'a aucun problème de collision.

Exemple d'un annuaire

Cas sans collision



Gestion des collisions

Par chaînage

- On gère un tableau de liste chaînées.

Gestion des collisions

Par chaînage

- On gère un tableau de liste chaînées.
- Les alvéoles ne sont plus des tuples (clé,valeur) mais des listes chaînées de tuples (clés,valeurs). Ces listes sont vides par défaut.

Gestion des collisions

Par chaînage

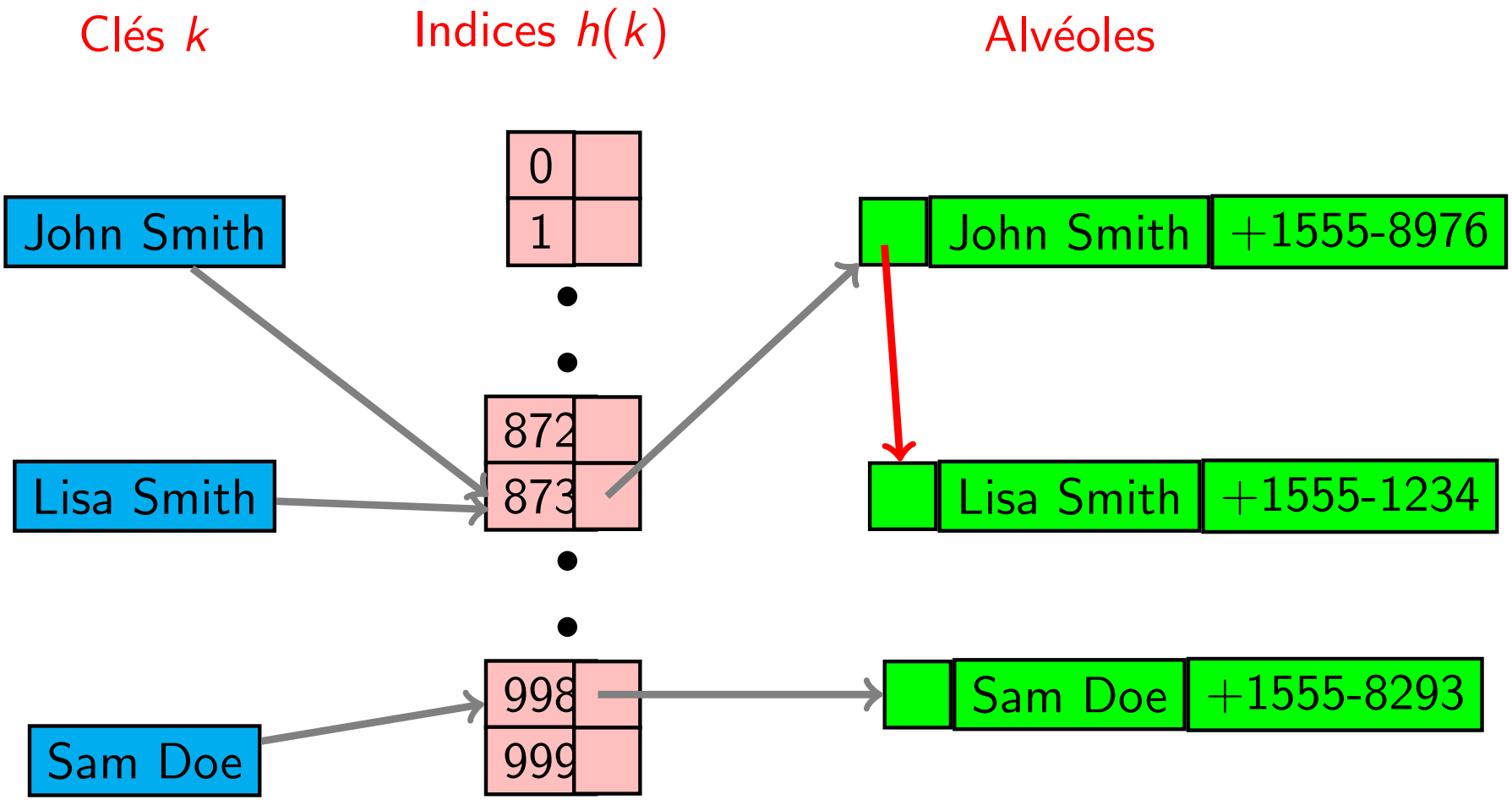
- On gère un tableau de liste chaînées.
- Les alvéoles ne sont plus des tuples (clé,valeur) mais des listes chaînées de tuples (clés,valeurs). Ces listes sont vides par défaut.
- Lors de l'ajout d'une nouvelle association comme (B,230), on cherche l'avéole à l'indice $h(B)$ du tableau des alvéoles. c'est une liste chaînée. On ajoute au bout de cette liste chaînée l'association (B,230).

Gestion des collisions

Par chaînage

- On gère un tableau de liste chaînées.
- Les alvéoles ne sont plus des tuples (clé,valeur) mais des listes chaînées de tuples (clés,valeurs). Ces listes sont vides par défaut.
- Lors de l'ajout d'une nouvelle association comme (B,230), on cherche l'avéole à l'indice $h(B)$ du tableau des alvéoles. c'est une liste chaînée. On ajoute au bout de cette liste chaînée l'association (B,230).
- Dans une recherche, le pire cas se produit quand la fonction de hachage associe un même indice à de nombreuses clés. Pour rechercher une clé particulière, il faut alors parcourir toute la liste chaînée de l'alvéole.

Gestion des collisions par chaînage



Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.

Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.
- Lorsqu'on insère l'association (k, v), on vérifie si la case $h(k)$ du tableau est libre.

Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.
- Lorsqu'on insère l'association (k, v), on vérifie si la case $h(k)$ du tableau est libre.
  - Si c'est le cas, tout va bien, on place l'association case $h(k)$.

Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.
- Lorsqu'on insère l'association (k, v), on vérifie si la case $h(k)$ du tableau est libre.
  - Si c'est le cas, tout va bien, on place l'association case $h(k)$.
  - Sinon, on part de la case k et on explore le tableau selon une certaine *fonction de sondage*.

Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.
- Lorsqu'on insère l'association (k, v), on vérifie si la case $h(k)$ du tableau est libre.
  - Si c'est le cas, tout va bien, on place l'association case $h(k)$.
  - Sinon, on part de la case k et on explore le tableau selon une certaine *fonction de sondage*.
    - La plus simple de ces fonctions consiste à exlorer d'abord la case $h(k) + 1$ puis la case $h(k) + 2$ etc. On l'appelle *sondage linéaire* (l'intervalle entre deux cases explorées est constant). Bien sûr, il faut travailler modulo le nombre de cases dans le tableau.

Gestion des collisions par adressage ouvert

- Les enregistrements (clés,valeurs) sont stockés dans un tableau.
- Lorsqu'on insère l'association (k, v), on vérifie si la case $h(k)$ du tableau est libre.
  - Si c'est le cas, tout va bien, on place l'association case $h(k)$.
  - Sinon, on part de la case k et on explore le tableau selon une certaine *fonction de sondage*.
    - La plus simple de ces fonctions consiste à exlorer d'abord la case $h(k) + 1$ puis la case $h(k) + 2$ etc. On l'appelle *sondage linéaire* (l'intervalle entre deux cases explorées est constant). Bien sûr, il faut travailler modulo le nombre de cases dans le tableau.
    - Autre méthode, le *sondage quadratique* : l'écart entre deux sondages augmente linéairement. Encore une fois, travailler modulo.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Insertion)

- On insère d'abord John Smith et son numéro. La valeur de hachage est 152. Pas de problème.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Insertion)

- On insère d'abord John Smith et son numéro. La valeur de hachage est 152. Pas de problème.
- On veut insérer ensuite Sandra Dee et son numéro. Malheureusement, la valeur de hachage est aussi 152 : il y a collision. On ajoute donc les coordonnées de Sandra à la première case libre après 152, soit 153.

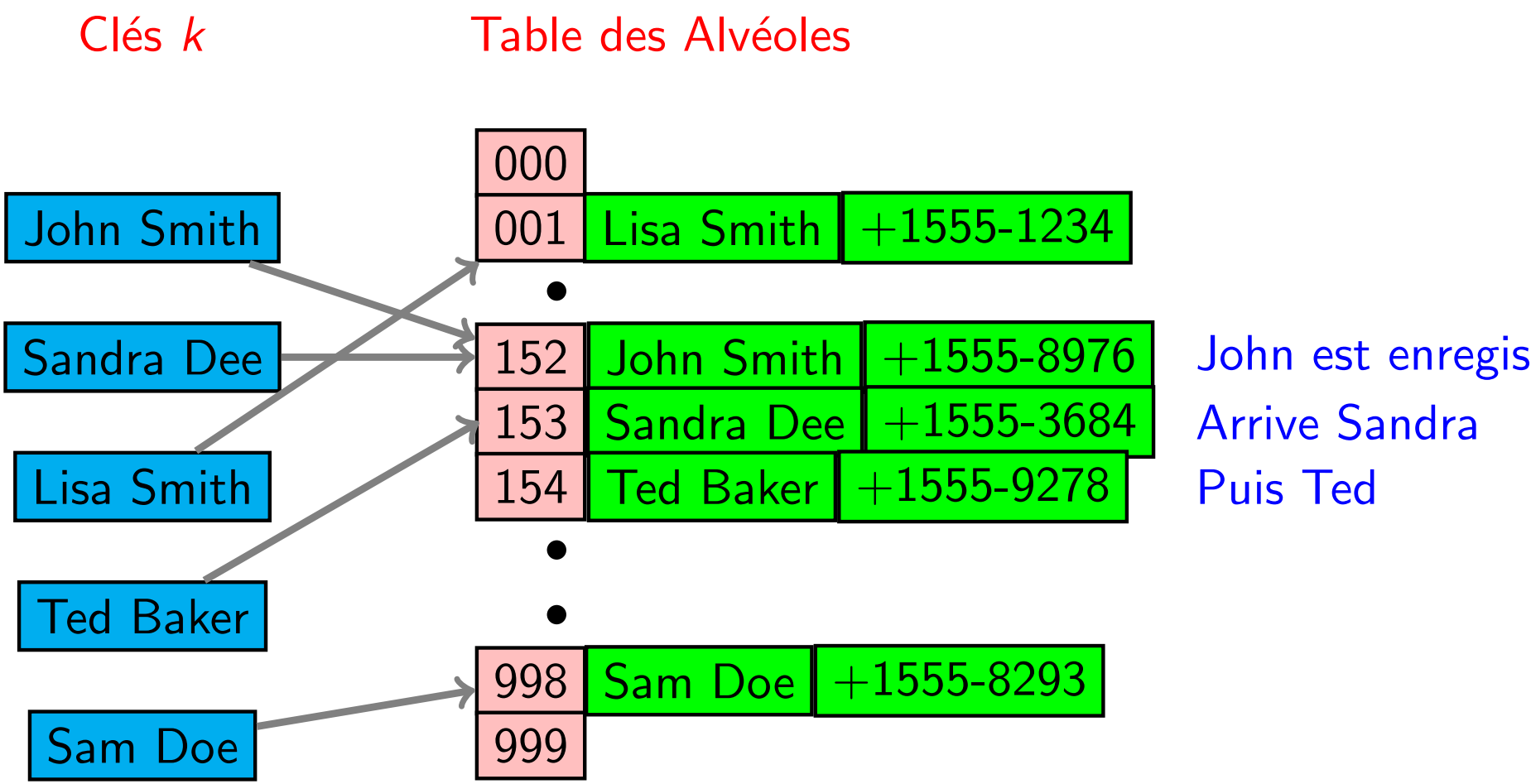
Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Insertion)

- On insère d'abord John Smith et son numéro. La valeur de hachage est 152. Pas de problème.
- On veut insérer ensuite Sandra Dee et son numéro. Malheureusement, la valeur de hachage est aussi 152 : il y a collision. On ajoute donc les coordonnées de Sandra à la première case libre après 152, soit 153.
- Arrive Ted Baker. La valeur de hachage associée est 153. cette valeur est bien unique mais la case correspondante est hélas occupée par Sandra (laquelle a été déplacée par rapport à sa valeur de hachage). On cherche donc pour Ted, la prochaine case vide, soit 154.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Wikipedia)



Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Recherche)

- On cherche les coordonnées de Ted. La valeur de hachage est 153.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Recherche)

- On cherche les coordonnées de Ted. La valeur de hachage est 153.
- En inspectant l'alvéole de la case 153, on se rend compte que la clé stockée est Sandra Dee et non Ted Baker.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Recherche)

- On cherche les coordonnées de Ted. La valeur de hachage est 153.
- En inspectant l'alvéole de la case 153, on se rend compte que la clé stockée est Sandra Dee et non Ted Baker.
- On examine donc la prochaine case selon le sondage linéaire (ici, case courante + 1). cette fois-ci, la clé stockée est bien Ted Baker. On renvoie son numéro.

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Recherche)

- On cherche les coordonnées de Ted. La valeur de hachage est 153.
- En inspectant l'alvéole de la case 153, on se rend compte que la clé stockée est Sandra Dee et non Ted Baker.
- On examine donc la prochaine case selon le sondage linéaire (ici, case courante + 1). cette fois-ci, la clé stockée est bien Ted Baker. On renvoie son numéro.
- En procédant ainsi, on peut arriver au bout du tableau sans trouver Ted Baker : il faut alors repartir du début (travailler modulo).

Gestion des collisions par adressage ouvert

Cas d'un sondage linéaire (Recherche)

- On cherche les coordonnées de Ted. La valeur de hachage est 153.
- En inspectant l'alvéole de la case 153, on se rend compte que la clé stockée est Sandra Dee et non Ted Baker.
- On examine donc la prochaine case selon le sondage linéaire (ici, case courante + 1). cette fois-ci, la clé stockée est bien Ted Baker. On renvoie son numéro.
- En procédant ainsi, on peut arriver au bout du tableau sans trouver Ted Baker : il faut alors repartir du début (travailler modulo).
- Avec le sondage linéaire, on s'arrête dès qu'on tombe sur une case vide (signe que Ted Baker n'est pas une clé).

Choix d'une bonne fonction de hachage

- Pour de bonnes performances il faut trouver un compromis entre

Choix d'une bonne fonction de hachage

- Pour de bonnes performances il faut trouver un compromis entre
  - la rapidité du calcul du hachage

Choix d'une bonne fonction de hachage

- Pour de bonnes performances il faut trouver un compromis entre
  - la rapidité du calcul du hachage
  - La taille à réserver pour l'espace de hachage,

Choix d'une bonne fonction de hachage

- Pour de bonnes performances il faut trouver un compromis entre
  - la rapidité du calcul du hachage
  - La taille à réserver pour l'espace de hachage,
  - La réduction du risque de collisions

Choix d'une bonne fonction de hachage

- Pour de bonnes performances il faut trouver un compromis entre
  - la rapidité du calcul du hachage
  - La taille à réserver pour l'espace de hachage,
  - La réduction du risque de collisions
- Prendre un nombre premier comme taille de table de hachage évite les problèmes de diviseurs communs et donc de collisions. Prendre une puissance de 2 comme taille permet un calcul modulo rapide.

Choix d'une bonne fonction de hachage

Il y a *grumelage* quand les valeurs de hachage se retrouvent côte à côte dans la table (cf exemple avec Ted Baker). Pour l'éviter :

- Privilégier les fonctions de hachage avec une distribution uniforme des valeurs de hachage.

Choix d'une bonne fonction de hachage

Il y a *grumelage* quand les valeurs de hachage se retrouvent côte à côte dans la table (cf exemple avec Ted Baker). Pour l'éviter :

- Privilégier les fonctions de hachage avec une distribution uniforme des valeurs de hachage.
- Le rapport $\mathrm{fc} = \frac{n}{k}$ où n est le nombre de tuples clés-valeurs et k la capacité du tableau est appelé *facteur de compression*. Si $\mathrm{fc} > \frac{1}{3}$, le risque de collision augmente.

Choix d'une bonne fonction de hachage

Il y a *grumelage* quand les valeurs de hachage se retrouvent côte à côte dans la table (cf exemple avec Ted Baker). Pour l'éviter :

- Privilégier les fonctions de hachage avec une distribution uniforme des valeurs de hachage.
- Le rapport $\mathrm{fc} = \frac{n}{k}$ où n est le nombre de tuples clés-valeurs et k la capacité du tableau est appelé *facteur de compression*. Si $\mathrm{fc} > \frac{1}{3}$, le risque de collision augmente.
- En `Python`, redimensionnement du tableau dès que le facteur de compression dépasse 2/3. La capacité est alors multipliée par 2.

Choix d'une bonne fonction de hachage

Il y a *grumelage* quand les valeurs de hachage se retrouvent côte à côte dans la table (cf exemple avec Ted Baker). Pour l'éviter :

- Privilégier les fonctions de hachage avec une distribution uniforme des valeurs de hachage.
- Le rapport $\mathrm{fc} = \frac{n}{k}$ où n est le nombre de tuples clés-valeurs et k la capacité du tableau est appelé *facteur de compression*. Si $\mathrm{fc} > \frac{1}{3}$, le risque de collision augmente.
- En `Python`, redimensionnement du tableau dès que le facteur de compression dépasse 2/3. La capacité est alors multipliée par 2.
- En OCaml, le redimensionnement intervient lorsque la moitié de la taille prévue à la création est atteinte (source **hashtbl.ml**)

Création

Le module `Hashtbl` contient tout ce qu'il faut.

- Création avec

```
create : ?random:bool -> int -> ('a, 'b) t
```

Création

Le module `Hashtbl` contient tout ce qu'il faut.

- Création avec

```
create : ?random:bool -> int -> ('a, 'b) t
```

- Création avec un tableau sous-jacent de taille 5 :

```
1 # Hashtbl.create 5;;
2 - : ('_weak1, '_weak2) Hashtbl.t = <abstr>
```

Création

Le module `Hashtbl` contient tout ce qu'il faut.

- Création avec
  `create : ?random:bool -> int -> ('a, 'b) t`
- Création avec un tableau sous-jacent de taille 5 :

```
1 # Hashtbl.create 5;;
2 - : ('_weak1, '_weak2) Hashtbl.t = <abstr>
```

- Le paramètre optionnel `random` permet de faire un choix aléatoire de la fonction de hachage.

Création

Le module `Hashtbl` contient tout ce qu'il faut.

- Création avec
  `create : ?random:bool -> int -> ('a, 'b) t`
- Création avec un tableau sous-jacent de taille 5 :

```
1 # Hashtbl.create 5;;
2 - : ('_weak1, '_weak2) Hashtbl.t = <abstr>
```

- Le paramètre optionnel `random` permet de faire un choix aléatoire de la fonction de hachage.
- Sans ce paramètre (`false` par défaut) la fonction de hachage est toujours la même, ce qui peut conduire une personne malveillante à effectuer une attaque par *déni de service* : le pirate se débrouille pour générer de nombreuse collisions, ce qui ralenti l'application considérablement.

Création

Le module `Hashtbl` contient tout ce qu'il faut.

- Création avec
  `create : ?random:bool -> int -> ('a, 'b) t`
- Création avec un tableau sous-jacent de taille 5 :

```
# Hashtbl.create 5;;
- : ('_weak1, '_weak2) Hashtbl.t = <abstr>
```

- Le paramètre optionnel `random` permet de faire un choix aléatoire de la fonction de hachage.
- Sans ce paramètre (`false` par défaut) la fonction de hachage est toujours la même, ce qui peut conduire une personne malveillante à effectuer une attaque par *déni de service* : le pirate se débrouille pour générer de nombreuse collisions, ce qui ralenti l'application considérablement.
- Nous n'utiliserons pas le paramètre optionnel en MP2I.

Ajout/recherche d'associations

- La fonction `replace : ('a, 'b) t -> 'a -> 'b -> unit` ajoute une association. Ainsi `replace tbl key data` ajoute l'alvéole `(key,data)`. Les anciennes associations de même clé sont effacées.

```
1 Hashtbl.replace d "toto" 1;;
2 Hashtbl.replace d "titi" 2;;
```

Ajout/recherche d'associations

- La fonction `replace : ('a, 'b) t -> 'a -> 'b -> unit` ajoute une association. Ainsi `replace tbl key data` ajoute l'alvéole `(key,data)`. Les anciennes associations de même clé sont effacées.

```
1 Hashtbl.replace d "toto" 1;;
2 Hashtbl.replace d "titi" 2;;
```

- La fonction `find : ('a, 'b) t -> 'a -> 'b` trouve la valeur associée à une clé. Si la clé n'est pas présente, une exception `Not_found` est soulevée.

```
1 # HashtblHashtbl.find d "toto";;
2 - : int = 5
3 # Hashtbl.find d "tata";;
4 Exception: Not_found.
```

Parcours des clés

- Avec la fonction
  `iter : ('a -> 'b -> unit) -> ('a, 'b) t -> unit`,
  on parcourt toutes les clés.

Parcours des clés

- Avec la fonction `iter : ('a -> 'b -> unit) -> ('a, 'b) t -> unit`, on parcourt toutes les clés.
- Exemple :

```
# Hashtbl.iter
    (fun k v -> Printf.printf "%s:%d\n" k v)
    d;;
toto:1
titi:2
- : unit = ()
```

Work in progress.