# STRUCTURES DE DONNEES ET ALGORITHMES

Esial 1ère année

Année 2006-2007

# Thèmes abordés

- Type Ensemble
- Type Table
- Notion de Hachage
- Implantation du Hachage

# Type Ensemble

# Quelques définitions

- L'ensemble S comprend tous les éléments x tels que *x appartient à S*
- L'ensemble vide ne contient pas d 'élément et est noté ∅
- Il n'y a pas d'ordre sur les éléments
- Un élément ne peut pas apparaître deux fois (sinon on parle de multi-ensemble)

# Utilisation d'ensembles

- Les ensembles permettent de représenter un collection d'objets où l'ordre entre les objets n'a pas d'importance :
  - Pièces d'un puzzle
  - Collection de BD
  - Registres non utilisés
  - Places d'un parking
  - …

# Opérations

- ajout d'un élément : $S \cup \{a\}$
- retrait d'un élément : $S \setminus \{a\}$
- union de deux ensembles : $S \cup T$
- intersection entre deux ensembles: $S \cap T$
- différence entre deux ensembles : $S \setminus T$
- Appartenance d'un élément : $x \in S$
- …

# Opérations sur les ensembles

Sorte Ensemble

Opérations

| | |
|---|---|
| ens_vide | : → Ensemble |
| appartient | : Ensemble x Elément → Booléen |
| supprimer | : Ensemble x Elément → Ensemble |
| ajouter | : Ensemble x Elément → Ensemble |
| taille | : Ensemble → Entier |
| est_vide | : Ensemble → Booléen |
| union | : Ensemble x Ensemble → Ensemble |
| inter | : Ensemble x Ensemble → Ensemble |
| diff | : Ensemble x Ensemble → Ensemble |

# Axiomatisation

- Constructeurs

  ens_vide : → Ensemble

  ajouter : Ensemble x Elément → Ensemble

- Définition des fonctions

  appartient(ens_vide,e) = faux

  appartient(ajouter(R,e),e') = (e=e') ou appartient(R,e')

  supprimer(ajouter(R,e),e) = R

  supprimer(ajouter(R,e),e') = ajouter(supprimer(R,e'),e)

  si e ≠ e'

- Exercice : Définir les autres opérations du type Ensemble

# Implantation

- Comment représenter un ensemble ?

  par un tableau ?

  par une liste ?

  par une pile ?

  par une file ?

  par autre chose ?

# Première approche

- Par une liste
  - L'ensemble {pomme, poire, banane}
  - peut être représenté par la liste :
  - [pomme, poire, banane]
- Dans ce cas, comment vont être implantées les opérations sur les ensembles ?
- Quel est l'impact sur les performances ?

# Implantation par une liste (1)

- Taille d'un ensemble
  - Taille de la liste
  - Temps constant dans le cas de ArrayList et LinkedList
- Appartient
  - 0 ≤ IndexOf(element) ≤ length()
  - Recherche linéaire

# Implantation par une liste (2)

- Ajout / Suppression
  - Il faut regarder si l'élément est déjà dans la liste (linéaire)
  - L'ajout peut se faire en queue (constant, sauf en cas de re-dimensionnement du tableau)
  - Le coût de la suppression dépend de l'implantation de la liste :
    - Tableau : il faut décaler les éléments
    - Liste simplement chaînée : il faut avoir accès a la cellule qui précède (il faut donc la mémoriser pendant l'étape de recherche)
    - Liste doublement chaînée : suppression classique

# Implantation par une liste (3)

- Union
  - Il faut ajouter tous les éléments d'un ensemble dans l'autre
  - Cela coûte cher parce que pour chaque élément ajouté, il faut regarder s'il n'existe pas déjà (temps en n*m)

# Résumé

- L'implantation d'un ensemble par une liste est faisable
  - Les opérations sont assez coûteuses
- L'implantation d'un multi-ensemble se fait mieux:
  - Les éléments peuvent apparaître plusieurs fois
- Dans tous les cas, ce n'est pas très satisfaisant lorsqu'on manipule de grands ensembles

# Remarques

- Il n'y a pas de notion d'ordre
- On peut ré-ordonner les éléments
- On peut ainsi utiliser une liste triée

# Quel est l'intérêt ?

- La recherche est plus efficace
  - Recherche dichotomique par exemple
- Suppression plus efficace
  - Parce que la recherche est plus rapide
- L’union se fait en temps linéaire
  - Fusion de listes
- L’ajout coûte un peu plus cher
  - Ajout ordonné

# Union : fusion de listes triées

(1,2,4,6,8) ∪ (7,9,10,11,12)

= (1, (2,4,6,8) ∪ (7,9,10,11,12))

= (1,2, (4,6,8) ∪ (7,9,10,11,12))

= … = (1,2,4,6, (8) ∪ (7,9,10,11,12))

= (1,2,4,6,7, (8) ∪ (9,10,11,12))

= (1,2,4,6,7,8, ∅ ∪ (9,10,11,12))

= (1,2,4,6,7,8,9,10,11,12)

# Autre représentation possible

Vecteur caractéristique

# Vecteur caractéristique

- Lorsqu'on manipule des sous-ensembles d'un ensemble fini U
- On peut associer à chaque élément de U une position i
- Un sous-ensemble de U peut se représenter par une suite de booléens indiquant si le i-ème élément est présent ou non

# Exemple

- Jeu de cartes : Pique Cœur Carreau Trèfle
- 1, 2,…, 9, 10, Valet, Dame, Roi
- position(1 Pique) = 1
- position(Valet Pique) = 11
- position(Valet Cœur) = 13 + 11 = 24
- position(Roi Trèfle) = 3*13 + 13 = 52
- Carré de Valet :

00000000001000000000001000000000001000000000100

# Opérations

- ajouter : mettre le i-ème bit à 1
- supprimer : mettre le i-ème bit à 0
- union : OU bit-à-bit
- intersection : ET bit-à-bit
- complément : NON bit-à-bit
- différence : intersection avec le complément

  diff(R,S) = R ET (NON(S))

# Y-a-t-il une dame dans le jeu ?

1000000001000000000010000000000000001000010000000100

(1 Pique, 10 Pique, 8 Cœur, … Carreau, ... Trèfle)

représentation d ’une dame : 0000000000010

4 dames

0000000000010000000000010000000000010000000000010

Aide : l ’appartenance se fait avec l ’intersection

1000000001000000000010000000000000001000010000000100

ET bit-à-bit

0000000000010000000000010000000000010000000000010

0000000000000000000000000000000010000000000000000

ensemble non vide : OUI

# Remarque

- Les vecteurs caractéristiques ne sont à utiliser que pour les ensembles finis de petite taille
- Cela ne fonctionne pas pour les multi-ensembles

- Existe-t-il une autre représentation ?
  - Réponse à la fin du cours

# Type Table

Associer un élément à un autre

(Map)

# Exemples

- mot → définition
- équipe → ensemble de joueurs
- ville → (destination, prix)
- numéro de CB → compte bancaire
- …

A une clé on associe un élément

# Opérations sur les dictionnaires

Sorte Table

Opérations

| | |
|---|---|
| table_vide | : → Table |
| appartient | : Table x Clé → Booléen |
| valeur | : Table x Clé → Elément |
| supprimer | : Table x Clé → Table |
| ajouter | : Table x Clé x Elément → Table |
| taille | : Table → Entier |
| est_vide | : Table → Booléen |

# Question

- Quelle structure utiliser pour mémoriser les couples :

Pomme → 150 kJ/100g

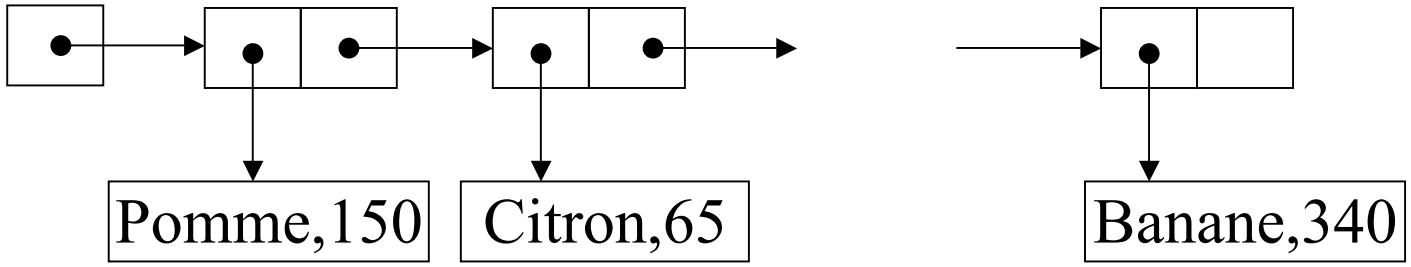
Citron → 65 kJ/100g

Orange → 115 kJ/100g

Banane → 340 kJ/100g

# Implantation par une liste

- liste de couples (clé,valeur)



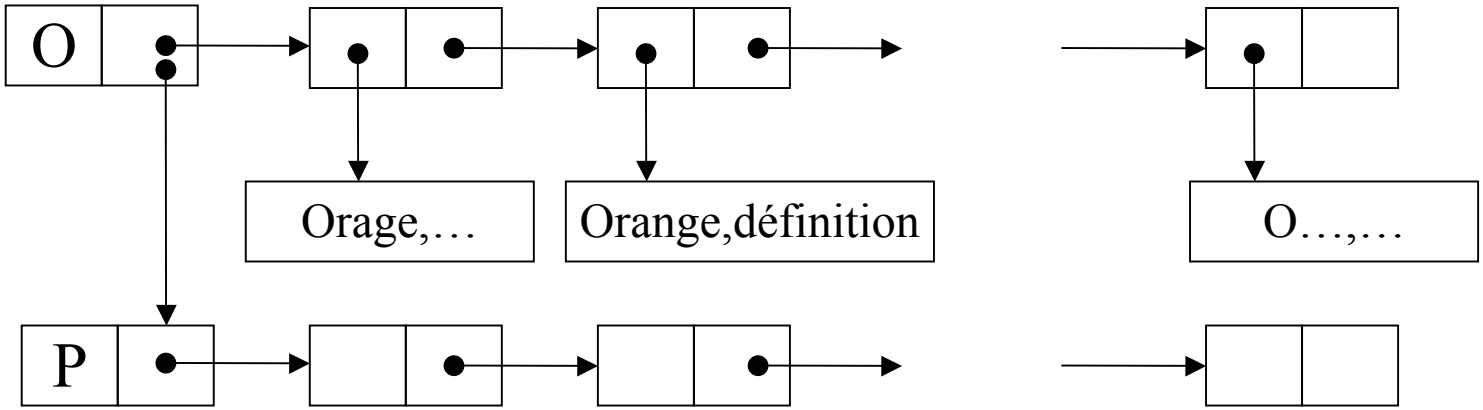

  - appartient, valeur : recherche séquentielle
  - supprimer : recherche séquentielle de l'élément à supprimer
  - ajouter : vérification de non présence (séquentielle)
- On peut éventuellement trier la liste

# Résumé

- On n’a pas tellement avancé
- Pour un ensemble
  - On ne sait pas faire mieux que les listes, éventuellement triées
  - Sauf dans le cas d’ensemble de petite taille
- Pour un dictionnaire
  - On se ramène à l’utilisation d’un ensemble de couples

# Question

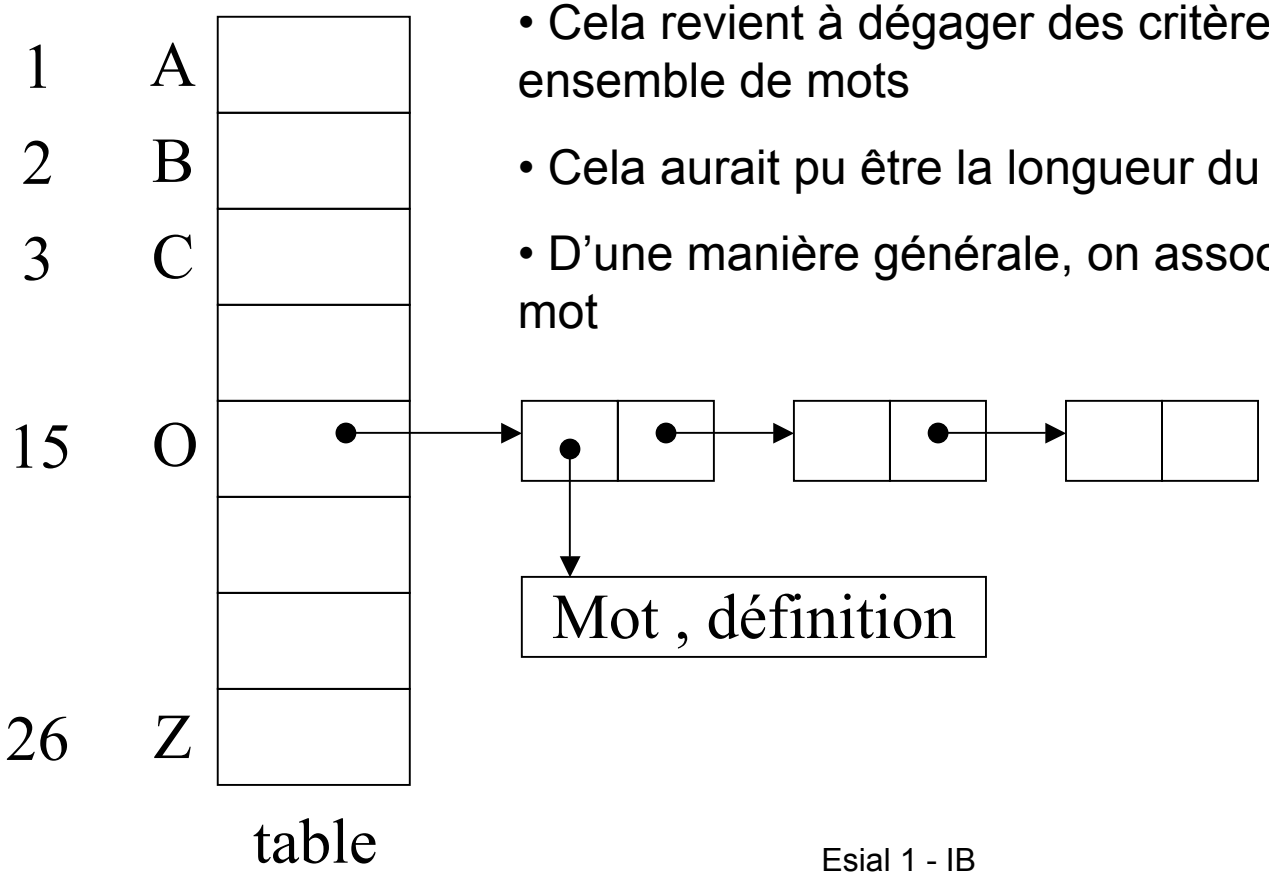
- Comment fait-on une recherche dans un dictionnaire ?
- Par exemple, la définition de « Orange »
- On cherche la première lettre de la clé



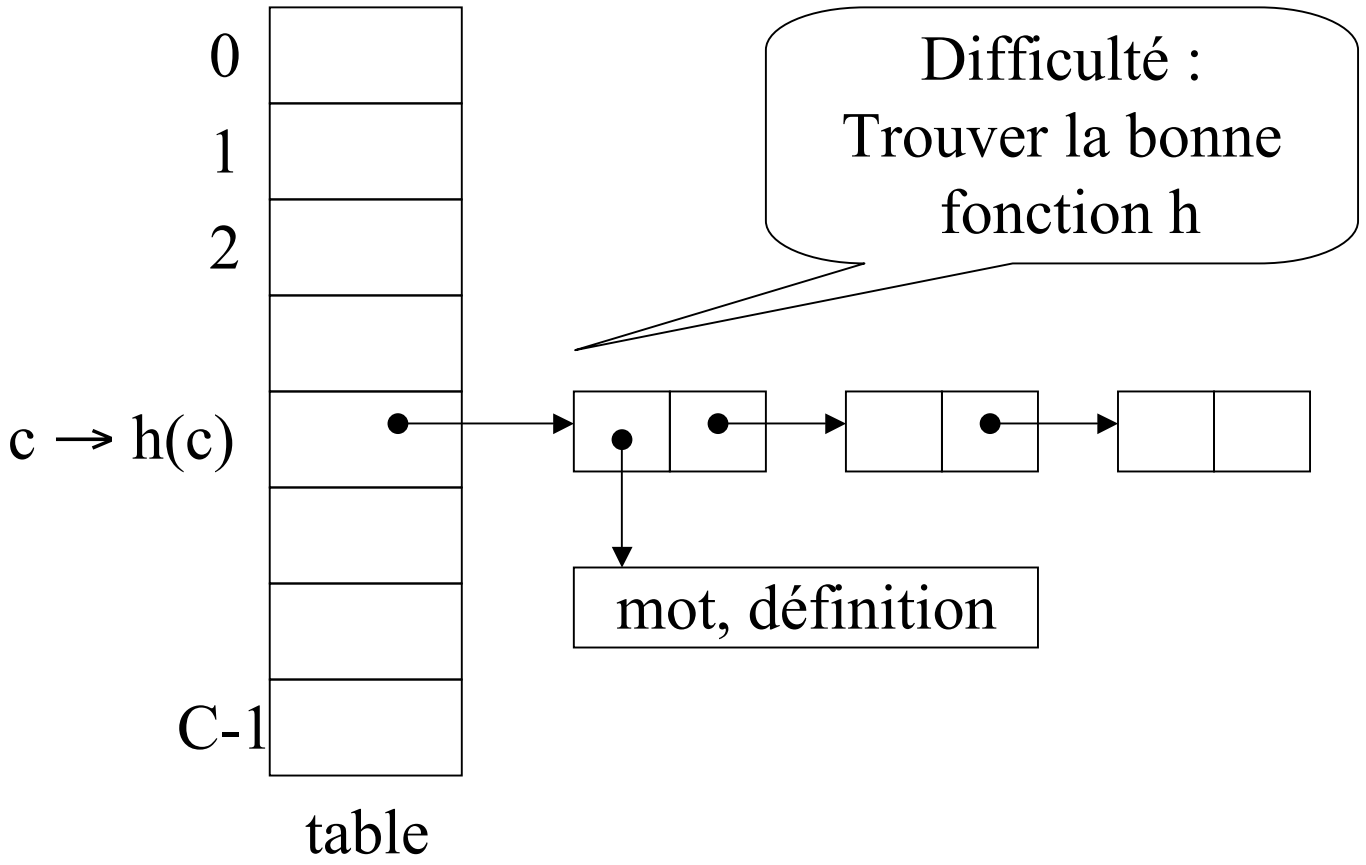

- Optimisation : accès direct à la lettre

# Idée

- On regroupe les mots commençant par la même lettre
- Cela revient à dégager des critères communs à un ensemble de mots
- Cela aurait pu être la longueur du mot
- D'une manière générale, on associe un entier à chaque mot



# Notion de Hachage



# Intérêt du hachage

- Cela permet de réduire la taille des listes dans lesquelles on cherche
  - Exemple : le mots commençant par « O »
- Problème : les listes ne sont pas de même longueur
  - On va plus vite pour chercher un mot commençant par « Z »
- Solution : mieux répartir les mots, en utilisant un autre critère

# Exemples de fonction de hachage (1)

- Mot → numéro de la première lettre
  - Beaucoup de collisions
  - Au plus 26 catégories différentes
- Mot → nombre de lettres du mot
  - Mauvaise répartition
  - Ne peut pas aller au dessus de 25

# Exemples de fonction de hachage (2)

- Mot → somme des lettres, modulo un entier
  - h(Pomme) = (16+15+13+13+5) mod 20 = 62 mod 20 = 2
  - h(Orange) = (15+18+1+14+7+5) mod 20 = 60 mod 20 = 0
  - h(Banane) = (2+1+14+1+14+5) mod 20 = 37 mod 20 = 17

  - Permet une meilleure répartition
  - Permet de répartir en plus de 26 catégories
  - Mais il faut un nombre minimum de lettres
    - Pour répartir dans 1000 cases par exemple

# Hachage ouvert (appelé également externe)

- Pour répartir un ensemble D de données en C classes, on se donne :
  - un tableau [0, C-1]
  - une fonction h : D $\rightarrow$ [0, C-1]
  - des listes pour résoudre les collisions
- Pour avoir une bonne répartition

  $\forall c \in [0,C-1]$,

  $card(\{x \mid x \in D \text{ et } h(x)=c\}) \approx card(D)/C$

  (i.e : *la plupart du temps* : $x1 \neq x2 \Rightarrow h(x1) \neq h(x2)$)

# Question

- Peut-on utiliser n'importe quel type de clé ?
  - OUI, mais il faut savoir hacher des objets
  - La classe doit implanter la méthode hashCode()

```
hash(x : Objet) = r : Entier
interne ← 0
pourtout champ
  interne ← ⊗ (interne , hash(x.champ))
  mixer(interne)
finpour
r ← projection( interne )
```

# En Java

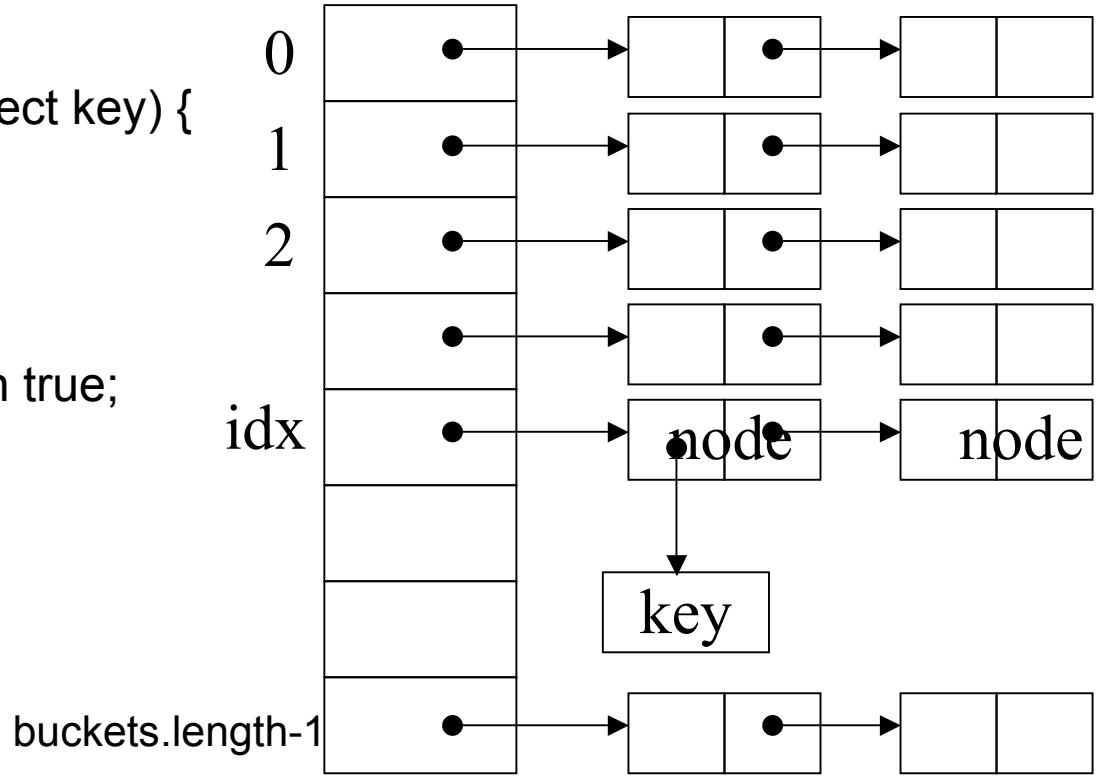
```
public class Object {
    public boolean equals(Object obj)
    public int hashCode()
    public String toString()
    protected Object clone()
    public final Class getClass()
}
```

# Une fonction de hachage simple (String)

```
public int hashCode() {
  if (cachedHashCode != 0) {
    return cachedHashCode;
  }
  int hashCode = 0;
  for (int i = 0; i < length(); i++) {
    hashCode = hashCode * 31 + value[i] + 1;
  }
  cachedHashCode = hashCode;
  return cachedHashCode;
}
```

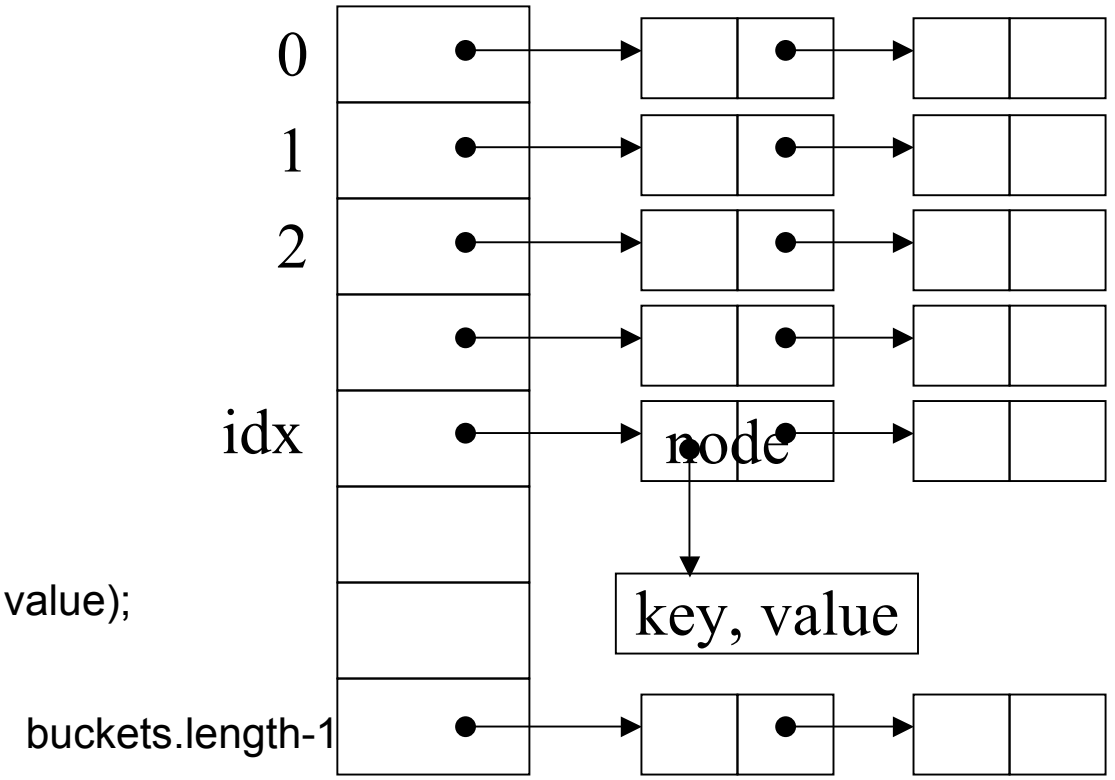
# Quelques algorithmes

```
final int hash(Object key) {
   return key == null ? 0 : Math.abs(key.hashCode() %
    buckets.length);
 }
public boolean containsKey(Object key) {
   int idx = hash(key);
   HashEntry e = buckets[idx];
   while (e != null) {
       if (equals(key, e.key)) return true;
       e = e.next;
   }
   return false;
 }
```



# Change some existing entry or add the new one

```
public Object put(Object key, Object value) {
  int idx = hash(key);
  HashEntry e = buckets[idx];
  while (e != null) {
    if (equals(key, e.key)) {
      Object r = e.value;
      e.value = value;
      return r;
    } else {
      e = e.next;
    }
  }
  HashEntry e = new HashEntry(key, value);
  e.next = buckets[idx];
  buckets[idx] = e;
  return null;
}
```



# Question

Que se passe-t-il lorsque la table est trop pleine ?

Cela va moins vite

Que faut-il faire ?

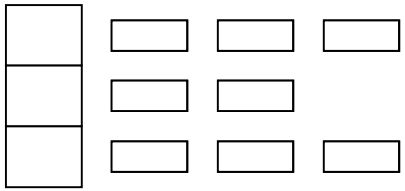
Agrandir la table

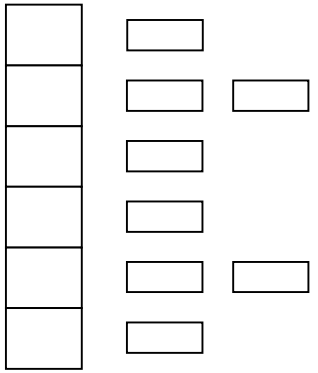
# rehash

```
private void rehash() {
   HashEntry[] oldBuckets = buckets;

   int newcapacity = (buckets.length * 2) + 1;
   threshold = (int) (newcapacity * loadFactor);
   buckets = new HashEntry[newcapacity];

   for (int i = oldBuckets.length - 1; i >= 0; i--) {
      HashEntry e = oldBuckets[i];
      while (e != null) {
         int idx = hash(e.key);
         HashEntry dest = buckets[idx];
         HashEntry next = e.next;
         e.next = buckets[idx];
         buckets[idx] = e;
         e = next;
       }
    }
}
```





# Secret d'une bonne implantation

- une bonne fonction de hachage
  - une fonction entraînant une bonne répartition
  - une fonction peu coûteuse

Rotation
hash(x : Objet) = r : Entier
interne ← 0
pourtout champ
  interne ← (interne<<4)^(interne>>28)^hash(x.champ))
finpour...
r ← interne mod prime (facultatif)

# Bilan (1)

- Quels sont les différentes façons de représenter une Table ?

  Liste
  Liste triée
  Table de Hachage

- Quelle est la meilleure ?

# Bilan (2)

- Quels sont les différentes façons de représenter un Ensemble ?

  Liste
  Vecteur Caractéristique
  Table de Hachage

# Java

```
public class HashSet {
…
 public boolean addAll(Collection c) {
    Iterator itr = c.iterator();
    boolean modified = false;
    int pos = c.size();
    while (--pos >= 0)
      modified |= add(itr.next());
    return modified;
  }
 public boolean add(Object o) {
    return map.put(o, "") == null;
  }
}
```

# Question

- Comment représenter un multi-ensemble ?

  {a,b,a,b,c}

- Peut-on représenter un ensemble (ou une table) par de la mémoire allouée statiquement ?
- Cela s'appelle le hachage interne (voir livres)

# The End