

Structures de Données

Les Tables

Olivier FESTOR

Telecom Nancy

26 Avril 2022

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

Représentation indexée

5. Table de hachage

Présentation

Implantation Java

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

Représentation indexée

5. Table de hachage

Présentation

Implantation Java

Contributeurs

Depuis la création du cours

- Martin Quinson (créateur)
- Adrien Kranenbuhl
- Sébastien da Silva
- Gérald Oster
- Olivier Festor

Vocabulaire

$$\mathcal{I} = ((k_1, v_1) , (k_2, v_2) , (k_3, v_3))$$

clé
↓
↑
valeur

Une table, aussi appelée dictionnaire, permet d'**associer** une **clé** à une **valeur**.

Exemple : k_1 est la **clé** associée à la **valeur** v_1 .

Associations courantes clé-valeur

- Mot → Définition
- Référence → Produit
- Pseudo → Profil

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

Représentation indexée

5. Table de hachage

Présentation

Implantation Java

Type Dictionary<K,V>

<i>empty</i> :		$\rightarrow \text{Dictionary}\langle K, V \rangle$
<i>has</i> :	$\text{Dictionary}\langle K, V \rangle \times K$	$\rightarrow \text{Boolean}$
<i>value</i> :	$\text{Dictionary}\langle K, V \rangle \times K$	$\rightarrow V$
<i>size</i> :	$\text{Dictionary}\langle K, V \rangle$	$\rightarrow \text{Integer}$
<i>isEmpty</i> :	$\text{Dictionary}\langle K, V \rangle$	$\rightarrow \text{Boolean}$
<i>add</i> :	$\text{Dictionary}\langle K, V \rangle \times K \times V$	$\rightarrow \text{Dictionary}\langle K, V \rangle$
<i>delete</i> :	$\text{Dictionary}\langle K, V \rangle \times K$	$\rightarrow \text{Dictionary}\langle K, V \rangle$

Type Dictionary<K,V>

empty : $\rightarrow \text{Dictionary}\langle K, V \rangle$

has : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow \text{Boolean}$

value : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow V$

size : $\text{Dictionary}\langle K, V \rangle \rightarrow \text{Integer}$

isEmpty : $\text{Dictionary}\langle K, V \rangle \rightarrow \text{Boolean}$

add : $\text{Dictionary}\langle K, V \rangle \times K \times V \rightarrow \text{Dictionary}\langle K, V \rangle$

delete : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow \text{Dictionary}\langle K, V \rangle$

Type Dictionary<K,V>

empty : $\rightarrow \text{Dictionary}\langle K, V \rangle$

has : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow \text{Boolean}$

value : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow V$

size : $\text{Dictionary}\langle K, V \rangle \rightarrow \text{Integer}$

isEmpty : $\text{Dictionary}\langle K, V \rangle \rightarrow \text{Boolean}$

add : $\text{Dictionary}\langle K, V \rangle \times K \times V \rightarrow \text{Dictionary}\langle K, V \rangle$

delete : $\text{Dictionary}\langle K, V \rangle \times K \rightarrow \text{Dictionary}\langle K, V \rangle$

Les préconditions

Les préconditions

$add(t, k, v)$ défini ssi

Les préconditions

$add(t, k, v)$ défini ssi $non\ has(t, k)$

Les préconditions

$add(t, k, v)$ défini ssi $non\ has(t, k)$
 $value(t, k)$ défini ssi

Les préconditions

$add(t, k, v)$ défini ssi $non\ has(t, k)$
 $value(t, k)$ défini ssi $has(t, k)$

Axiomes avec *has*

Axiomes avec *has*

$$has(empty(), k) =$$

Axiomes avec *has*

$$has(empty(), k) = faux$$

Axiomes avec *has*

$$has(empty(), k) = faux$$

$$has(add(t, k_1, v), k_2) =$$

Axiomes avec *has*

$$\begin{aligned} \text{has}(\text{empty}(), k) &= \text{faux} \\ \text{has}(\text{add}(t, k_1, v), k_2) &= \begin{cases} \text{vrai} & \text{si } k_1 = k_2 \\ \text{has}(t, k_2) & \text{sinon} \end{cases} \end{aligned}$$

Axiomes avec *has*

$$has(empty(), k) = faux$$

$$has(add(t, k_1, v), k_2) = \begin{cases} vrai & \text{si } k_1 = k_2 \\ has(t, k_2) & \text{sinon} \end{cases}$$

$$has(delete(t, k_1), k_2) =$$

Axiomes avec *has*

$$has(empty(), k) = faux$$

$$has(add(t, k_1, v), k_2) = \begin{cases} vrai & \text{si } k_1 = k_2 \\ has(t, k_2) & \text{sinon} \end{cases}$$

$$has(delete(t, k_1), k_2) = \begin{cases} faux & \text{si } k_1 = k_2 \\ has(t, k_2) & \text{sinon} \end{cases}$$

Axiomes avec *value*

Axiomes avec *value*

$$value(empty(), k) =$$

Axiomes avec *value*

$value(empty(), k) = \text{Violation de précondition !}$

Axiomes avec *value*

$value(empty(), k) =$ Violation de précondition !

$value(add(t, k_1, v), k_2) =$

Axiomes avec *value*

$value(empty(), k) =$ Violation de précondition !

$$value(add(t, k_1, v), k_2) = \begin{cases} v & \text{si } k_1 = k_2 \\ value(t, k_2) & \text{sinon} \end{cases}$$

Axiomes avec *value*

$$\begin{aligned} \text{value}(\text{empty}(), k) &= \text{Violation de précondition !} \\ \text{value}(\text{add}(t, k_1, v), k_2) &= \begin{cases} v & \text{si } k_1 = k_2 \\ \text{value}(t, k_2) & \text{sinon} \end{cases} \\ \text{value}(\text{delete}(t, k_1), k_2) &= \end{aligned}$$

Axiomes avec *value*

$value(empty(), k) =$ Violation de précondition !

$value(add(t, k_1, v), k_2) = \begin{cases} v & \text{si } k_1 = k_2 \\ value(t, k_2) & \text{sinon} \end{cases}$

$value(delete(t, k_1), k_2) = value(t, k_2) \text{ (si } k_1 \neq k_2)$

Axiomes avec *size*

Axiomes avec *size*

$$\text{size}(\text{empty}()) =$$

Axiomes avec *size*

$$\text{size}(\text{empty}()) = 0$$

Axiomes avec *size*

$$\begin{aligned} \text{size}(\text{empty}()) &= 0 \\ \text{size}(\text{add}(t, k, v)) &= \end{aligned}$$

Axiomes avec *size*

$$\text{size}(\text{empty}()) = 0$$

$$\text{size}(\text{add}(t, k, v)) = \text{size}(t) + 1$$

Axiomes avec *size*

$$\text{size}(\text{empty}()) = 0$$

$$\text{size}(\text{add}(t, k, v)) = \text{size}(t) + 1$$

$$\text{size}(\text{delete}(t, k)) =$$

Axiomes avec *size*

$$\begin{aligned} \text{size}(\text{empty}()) &= 0 \\ \text{size}(\text{add}(t, k, v)) &= \text{size}(t) + 1 \\ \text{size}(\text{delete}(t, k)) &= \begin{cases} \text{size}(t) - 1 & \text{si } \text{has}(t, k) \\ \text{size}(t) & \text{sinon} \end{cases} \end{aligned}$$

Axiomes avec *isEmpty*

Axiomes avec *isEmpty*

$$\text{isEmpty}(\text{empty}()) =$$

Axiomes avec *isEmpty*

$$\text{isEmpty}(\text{empty}()) = \text{vrai}$$

Axiomes avec *isEmpty*

$$\begin{aligned} \text{isEmpty}(\text{empty}()) &= \text{vrai} \\ \text{isEmpty}(\text{add}(t, k, v)) &= \end{aligned}$$

Axiomes avec *isEmpty*

$$\begin{aligned} \text{isEmpty}(\text{empty}()) &= \text{vrai} \\ \text{isEmpty}(\text{add}(t, k, v)) &= \text{faux} \end{aligned}$$

Axiomes avec *isEmpty*

$$\begin{aligned} \text{isEmpty}(\text{empty}()) &= \text{vrai} \\ \text{isEmpty}(\text{add}(t, k, v)) &= \text{faux} \\ \text{isEmpty}(\text{delete}(\text{empty}(), k)) &= \end{aligned}$$

Axiomes avec *isEmpty*

$$\begin{aligned} \text{isEmpty}(\text{empty}()) &= \text{vrai} \\ \text{isEmpty}(\text{add}(t, k, v)) &= \text{faux} \\ \text{isEmpty}(\text{delete}(\text{empty}(), k)) &= \text{vrai} \end{aligned}$$

Axiomes avec *isEmpty*

$$\text{isEmpty}(\text{empty}()) = \text{vrai}$$

$$\text{isEmpty}(\text{add}(t, k, v)) = \text{faux}$$

$$\text{isEmpty}(\text{delete}(\text{empty}(), k)) = \text{vrai}$$

$$\text{isEmpty}(\text{delete}(\text{add}(t, k_1, v), k_2)) =$$

Axiomes avec *isEmpty*

$$\text{isEmpty}(\text{empty}()) = \text{vrai}$$

$$\text{isEmpty}(\text{add}(t, k, v)) = \text{faux}$$

$$\text{isEmpty}(\text{delete}(\text{empty}(), k)) = \text{vrai}$$

$$\text{isEmpty}(\text{delete}(\text{add}(t, k_1, v), k_2)) = \begin{cases} \text{isEmpty}(t) & \text{si } k_1 = k_2 \\ \text{faux} & \text{sinon} \end{cases}$$

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

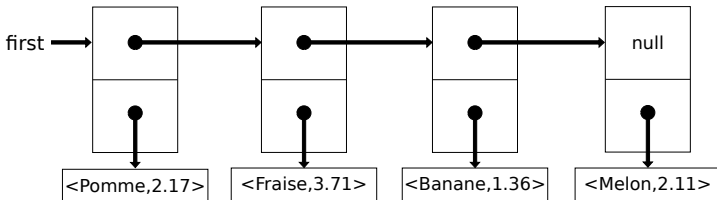
Représentation indexée

5. Table de hachage

Présentation

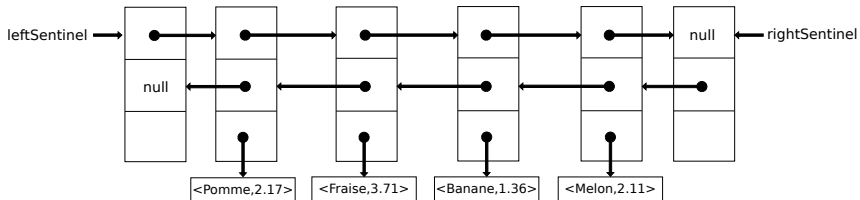
Implantation Java

Simple chaînage



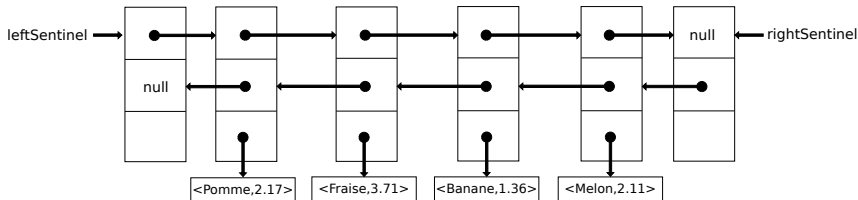
- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

Double chaînage



- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

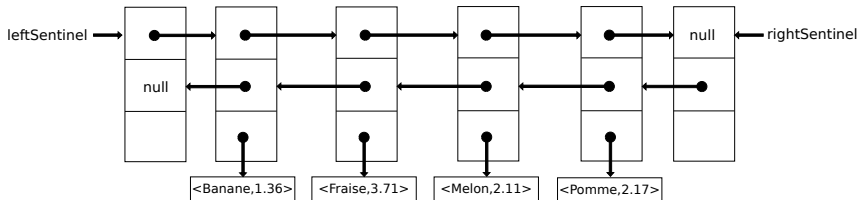
Double chaînage



- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

Et maintenant la liste triée + cursor ?

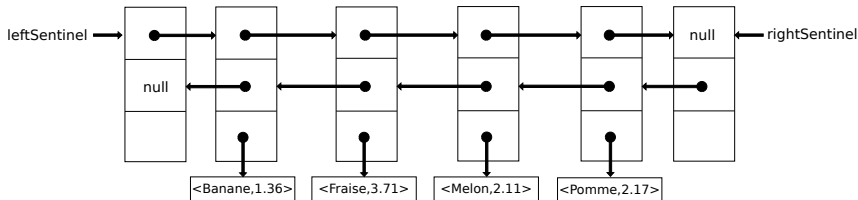
Double chaînage



- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

Et maintenant la liste triée + cursor ?

Double chaînage

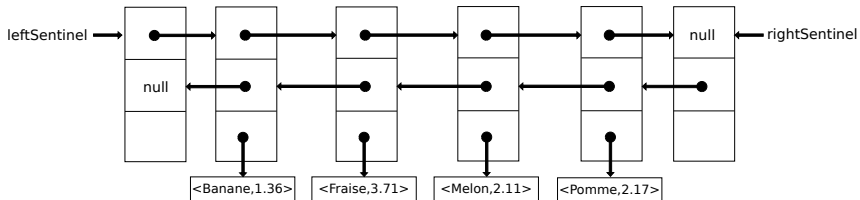


- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

Et maintenant la liste triée + cursor ?

- Ajout en $\Theta(N/2)$

Double chaînage

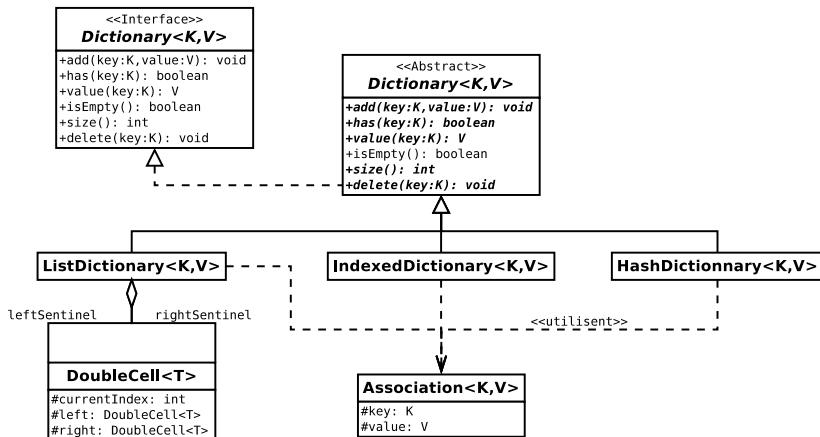


- Ajout en temps constant ($\Theta(1)$)
- Recherche en temps proportionnel à la taille de la liste ($\Theta(N)$)

Et maintenant la liste triée + cursor ?

- Ajout en $\Theta(N/2)$
- Recherche en $\Theta(N/2)$

Exercice 3.B



Écrivez les méthodes d'ajout et de suppression de la classe *ListDictionary<K, V>*.

Solution

```
1  public class ListDictionary<K,E>
2      extends AbstractDictionary<K,V>
3  {
4      protected DoubleLinkedList<Association<K,V>> elements;
5
6      public void add( K key, V value )
7      {
8          // À compléter...
9      }
10     public void delete( K key )
11     {
12         // À compléter...
13     }
14 }
```

Solution de *add()*

```
1  public class ListDictionary<K,E>
2              extends AbstractDictionary<K,V>
3  {
4      protected DoubleLinkedList<Association<K,V>> elements;
5
6      public void add( K key, V value )
7      {
8          assert( this.has(key) ) : "Precondition violee";
9          elements.addFirst(new Association(key,value));
10     }
11     ...
12 }
```

Solution

```
1  public class ListDictionary<K,E>
2          extends AbstractDictionary<K,V>
3  {
4      protected DoubleLinkedList<Association<K,V>> elements;
5
6      public void add( K key, V value )
7      {
8          // Ok !
9      }
10     public void delete( K key )
11     {
12         // À compléter...
13     }
14 }
```

Solution de *delete()*

```
1  public class ListDictionary<K,E>
2          extends AbstractDictionary<K,V>
3  {
4      protected DoubleLinkedList<Association<K,V>> elements;
5
6      ...
7      public void delete( K key )
8      {
9          boolean trouve = false;
10         int rang = 0;
11         while (!trouve && rang<elements.size())
12         {
13             trouve = elements.get(rang).key().equals(key);
14             if ( !trouve ) rang++;
15         }
16         if ( trouve ) elements.remove(rang);
17     }
18 }
```


Bof bof...

L'implantation chaînée :

- a exactement les mêmes performances que liste chaînées
- ne tire pas profit de l'ordre sur les clés

Bof bof...

L'implantation chaînée :

- a exactement les mêmes performances que liste chaînées
- ne tire pas profit de l'ordre sur les clés

Les tables deviennent de basiques listes chaînées d'associations.

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

Représentation indexée

5. Table de hachage

Présentation

Implantation Java

Qu'est-ce que diviser pour régner ?

Principe

Découper la table en sous-tables.

$$\mathcal{T} = (\left. \begin{array}{l} ((k_1, v_1) , (k_2, v_2) , (k_3, v_3)) ; \\ ((k_4, v_4) , (k_5, v_5)) ; \\ ((k_6, v_6) , (k_7, v_7) , (k_8, v_8) , (k_9, v_9)) \end{array} \right\} 3 \text{ sous-tables})$$

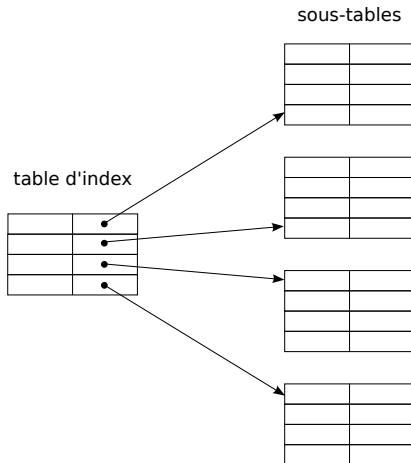
Conséquences :

1. On cherche d'abord la sous-table susceptible de contenir la clé
2. On cherche ensuite la clé dans la sous-table correspondante

Comment ça marche ?

Deux conditions nécessaires :

- Disposer d'un ensemble d'associations <clé,valeur>
- Avoir un ordre total sur les clés

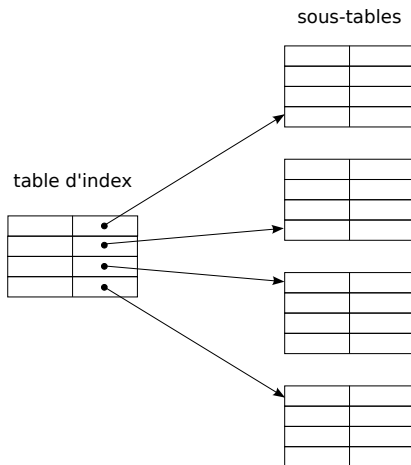


À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, compléter le diagramme de droite.

Précisions :

- La table d'index contient la dernière clé de chaque sous-table
- Le dernier index contient la plus grande des clés possibles pour préparer les insertions

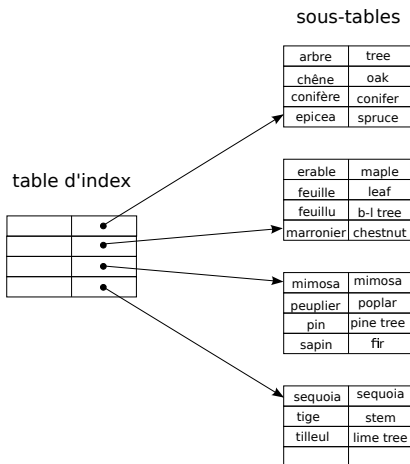


À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, compléter le diagramme de droite.

Précisions :

- La table d'index contient la dernière clé de chaque sous-table
- Le dernier index contient la plus grande des clés possibles pour préparer les insertions

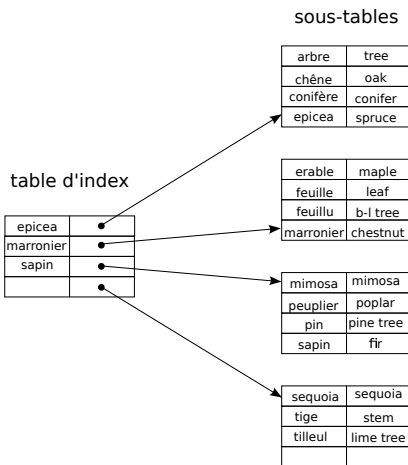


À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, compléter le diagramme de droite.

Précisions :

- La table d'index contient la dernière clé de chaque sous-table
- Le dernier index contient la plus grande des clés possibles pour préparer les insertions



À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, compléter le diagramme de droite.

Précisions :

- La table d'index contient la dernière clé de chaque sous-table
- Le dernier index contient la plus grande des clés possibles pour préparer les insertions

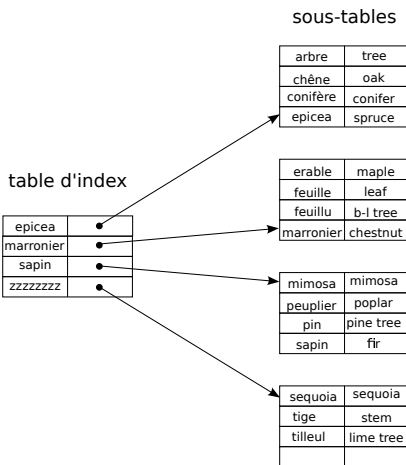


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

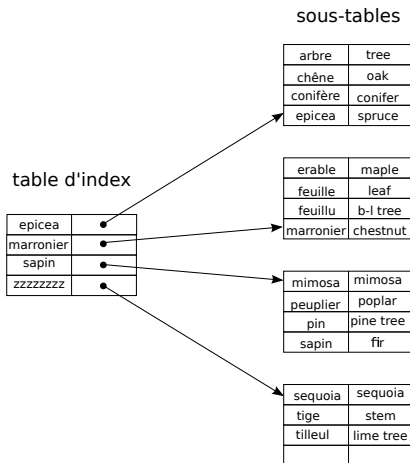


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?

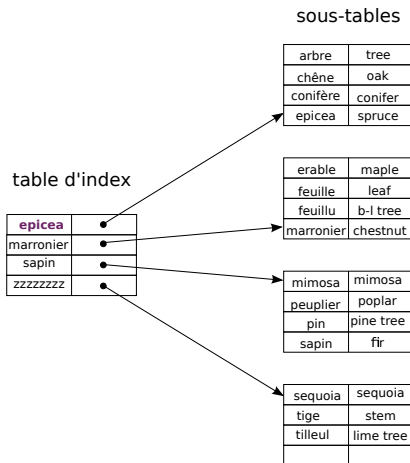


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?

Non → index suivant

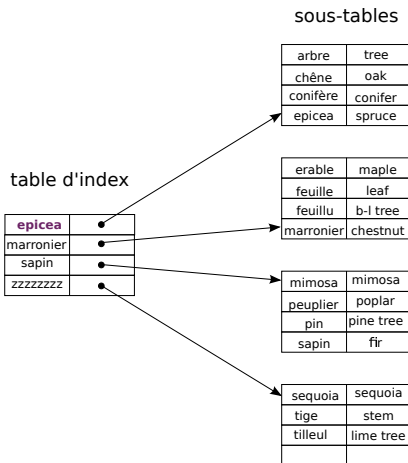


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?

Non → index suivant

2. "pin" < "marronnier" ?

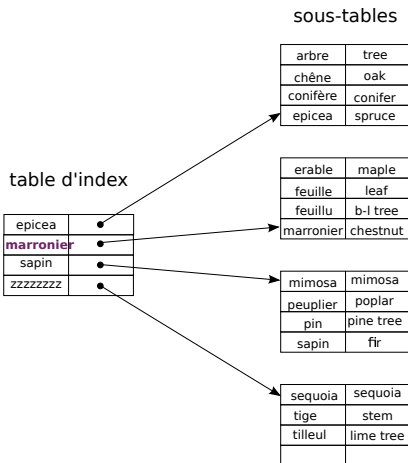


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant

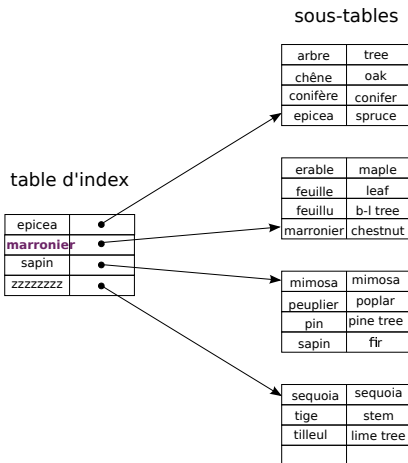


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?

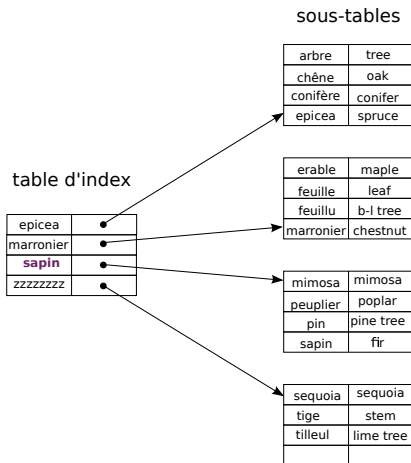


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui

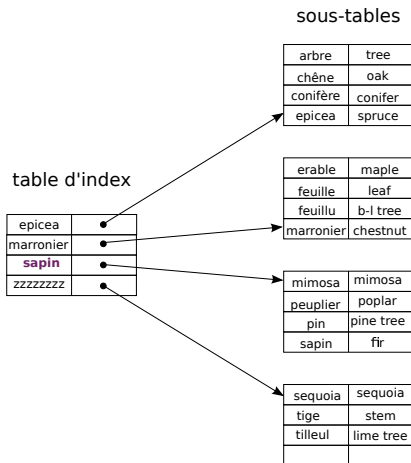


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?

Non → index suivant

2. "pin" < "marronnier" ?

Non → index suivant

3. "pin" < "sapin" ?

Oui → recherche dans la
sous-table correspondante

table d'index

epicea	•
marronnier	•
sapin	•
zzzzzzzz	•

sous-tables

arbre	tree
chêne	oak
conifère	conifer
epicea	spruce

erable	maple
feuille	leaf
feuillu	b-l tree
marronnier	chestnut

mimosa	mimosa
peuplier	poplar
pin	pine tree
sapin	fir

sequoia	sequoia
tige	stem
tilleul	lime tree

Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?

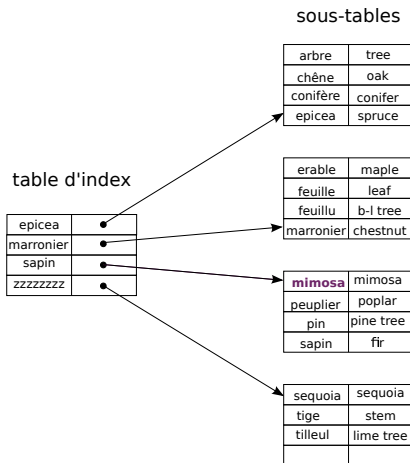


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante

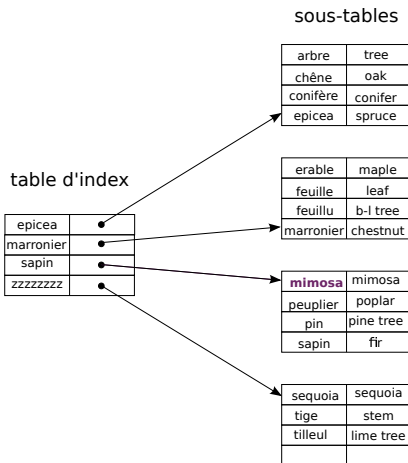


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante
5. "peuplier" == "pin" ?

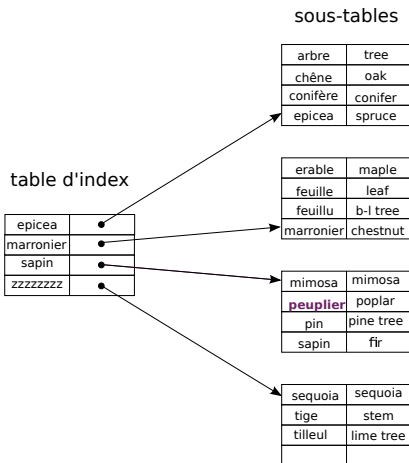


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante
5. "peuplier" == "pin" ?
Non → clé suivante

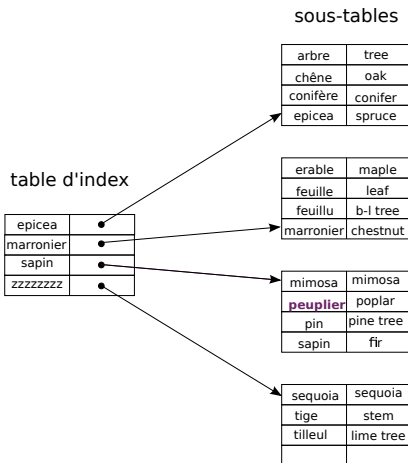


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante
5. "peuplier" == "pin" ?
Non → clé suivante
6. "pin" == "pin" ?

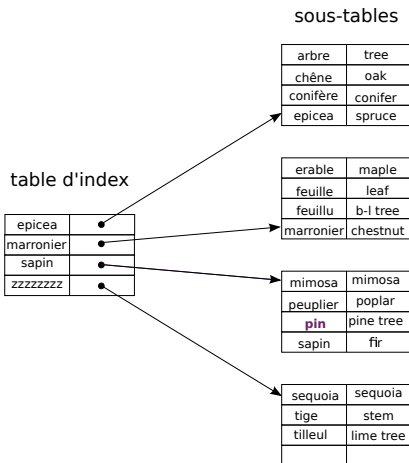


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante
5. "peuplier" == "pin" ?
Non → clé suivante
6. "pin" == "pin" ?
Oui

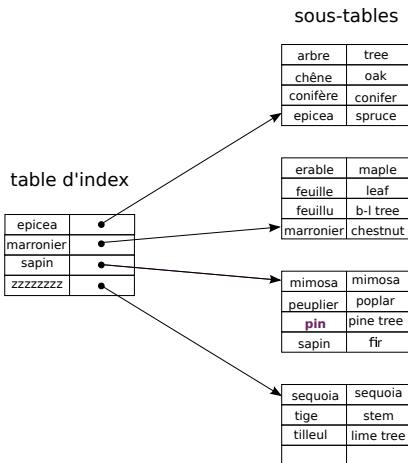
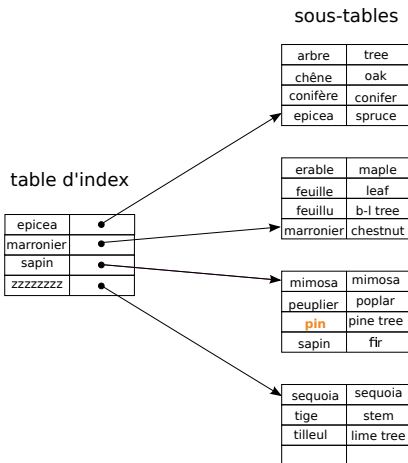


Illustration d'une recherche

La table contient-elle la clé
"pin" ?

1. "pin" < "epicea" ?
Non → index suivant
2. "pin" < "marronnier" ?
Non → index suivant
3. "pin" < "sapin" ?
Oui → recherche dans la sous-table correspondante
4. "mimosa" == "pin" ?
Non → clé suivante
5. "peuplier" == "pin" ?
Non → clé suivante
6. "pin" == "pin" ?
Oui → Clé trouvée !



Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Avec N éléments répartis en M sous-table.

Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Avec N éléments répartis en M sous-table.

Recherche en deux étape :

1. Recherche de l'index,
2. Recherche dans la sous-table.

Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Avec N éléments répartis en M sous-table.

Recherche en deux étape :

1. Recherche de l'index, \rightarrow **complexité en $\Theta(M)$**
2. Recherche dans la sous-table.

Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Avec N éléments répartis en M sous-table.

Recherche en deux étape :

1. Recherche de l'index, \rightarrow **complexité en $\Theta(M)$**
2. Recherche dans la sous-table. \rightarrow **complexité en $\Theta(N/M)$**

Complexité

Quelle est la complexité au pire cas d'une recherche dans une table indexée ?

Avec N éléments répartis en M sous-table.

Recherche en deux étape :

1. Recherche de l'index, \rightarrow **complexité en $\Theta(M)$**
2. Recherche dans la sous-table. \rightarrow **complexité en $\Theta(N/M)$**

La complexité d'une recherche est donc en $\Theta(M + N/M)$

Complexité minimale

Une idée pour améliorer la recherche ?

Complexité minimale

Une idée pour améliorer la recherche ?

On peut procéder par dichotomie.

Complexité minimale

Une idée pour améliorer la recherche ?

On peut procéder par dichotomie.

Recherche en deux étapes :

1. Recherche de l'index
2. Recherche dans la sous-table.

Complexité minimale

Une idée pour améliorer la recherche ?

On peut procéder par dichotomie.

Recherche en deux étapes :

1. Recherche de l'index \rightarrow **complexité en $\Theta(\log_2(M))$**
2. Recherche dans la sous-table.

Complexité minimale

Une idée pour améliorer la recherche ?

On peut procéder par dichotomie.

Recherche en deux étapes :

1. Recherche de l'index \rightarrow **complexité en $\Theta(\log_2(M))$**
2. Recherche dans la sous-table. \rightarrow **complexité en $\Theta(\log_2(\frac{N}{M}))$**

Complexité minimale

Une idée pour améliorer la recherche ?

On peut procéder par dichotomie.

Recherche en deux étapes :

1. Recherche de l'index \rightarrow **complexité en $\Theta(\log_2(M))$**
2. Recherche dans la sous-table. \rightarrow **complexité en $\Theta(\log_2(\frac{N}{M}))$**

La complexité d'une recherche est donc en
 $\Theta(\log_2(M) + \log_2(\frac{N}{M})) = \Theta(\log_2(N))$

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8
10000	100	200	13,3	15,1

Comparaison des complexités

N	M	Itératif $(M + N/M)$	Dichotomie $(\log_2(N))$	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8
10000	100	200	13,3	15,1
10000	1000	1010	13,3	76

Comparaison des complexités

N	M	Itératif ($M + N/M$)	Dichotomie ($\log_2(N)$)	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8
10000	100	200	13,3	15,1
10000	1000	1010	13,3	76
10000	5000	5002	13,3	376,4

Comparaison des complexités

N	M	Itératif ($M + N/M$)	Dichotomie ($\log_2(N)$)	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8
10000	100	200	13,3	15,1
10000	1000	1010	13,3	76
10000	5000	5002	13,3	376,4
1000000000	10000	110000	29,9	3679,3

Comparaison des complexités

N	M	Itératif ($M + N/M$)	Dichotomie ($\log_2(N)$)	Rapport
15	4	7,8	3,9	2
15	6	8,5	3,9	2,2
15	10	11,5	3,9	2,9
100	10	20	6,6	3
100	20	25	6,6	3,8
100	70	71,4	6,6	10,8
10000	100	200	13,3	15,1
10000	1000	1010	13,3	76
10000	5000	5002	13,3	376,4
1000000000	10000	110000	29,9	3679,3

La dichotomie, c'est génial !

Plan

1. Présentation

Vocabulaire

2. Spécification algébrique

Opérations

Préconditions

Axiomes

3. Implémentation chaînée

Présentation

Implantation Java

Bilan

4. Diviser pour régner

Présentation

Représentation indexée

5. Table de hachage

Présentation

Implantation Java

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Comment faire correspondre les deux structures ?

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Comment faire correspondre les deux structures ?

Tableau :

Dictionary :

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Comment faire correspondre les deux structures ?

Tableau : Indice → Valeur

Dictionary : Clé → Valeur

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Comment faire correspondre les deux structures ?

Tableau :	Indice	→	Valeur
	↑ ?		
Dictionary :	Clé	→	Valeur

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

→ Les tableaux

Comment faire correspondre les deux structures ?

Tableau :	Indice	→	Valeur
	↑ ?	Fonction de hachage !	
Dictionary :	Clé	→	Valeur

Pourquoi ?

Le problème/objectif

Pourrait-on trouver une implantation d type *Dictionary* avec une complexité de recherche en $\Theta(1)$?

Quelle structure de données connue a cette complexité ?

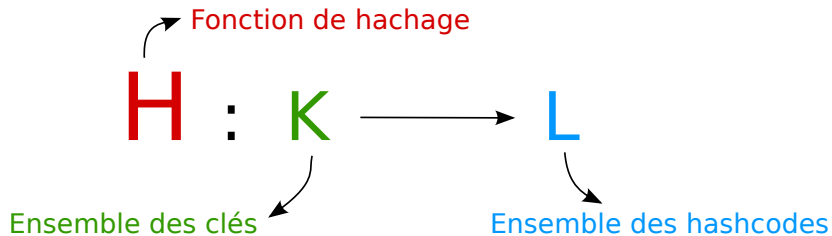
→ Les tableaux

Comment faire correspondre les deux structures ?

Tableau :	Indice	→	Valeur
	↑ ?		
Dictionary :	Clé	→	Valeur

Fonction de hachage !

Comment ça marche ?

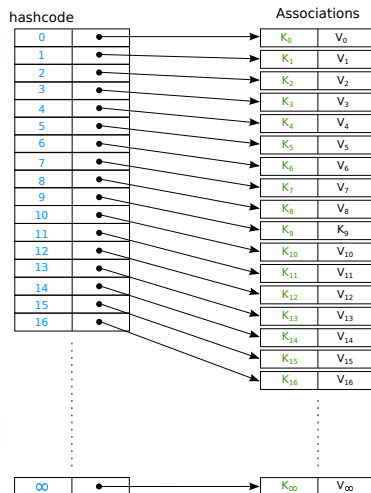


On utilise une **fonction de hachage** qui retourne un **hashcode** à partir d'une clé.

- Un hashcode = un indice dans la table d'index
- Une fonction de hachage **parfaite** retournerait un **hashcode unique** pour chaque clé

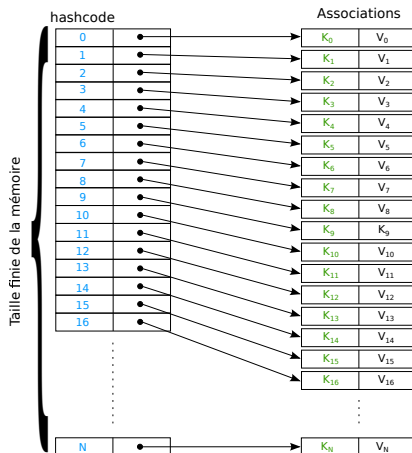
Limitations matérielles

En réalité, le nombre de clés possibles est souvent infini...



Limitations matérielles

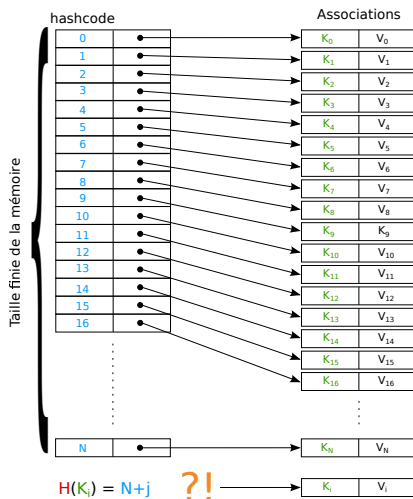
En réalité, le nombre de clés possibles est souvent infini... mais la mémoire est finie :-).



Limitations matérielles

En réalité, le nombre de clés possibles est souvent infini... mais la mémoire est finie :-).

**Clés infinies + mémoire finie
= Problème de collisions !**

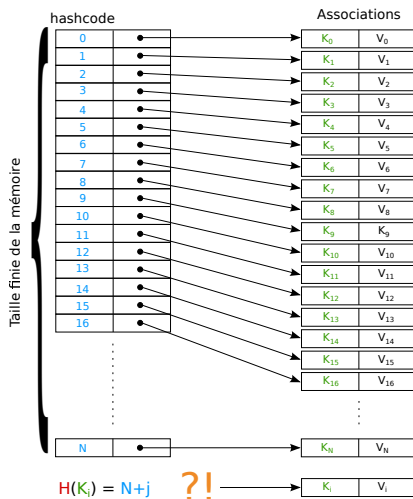


Limitations matérielles

En réalité, le nombre de clés possibles est souvent infini... mais la mémoire est finie :-).

**Clés infinies + mémoire finie
= Problème de collisions !**

Solution :



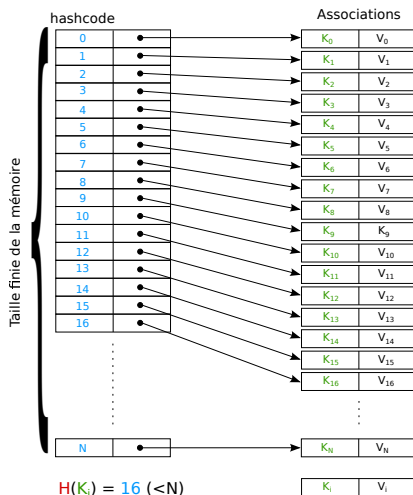
Limitations matérielles

En réalité, le nombre de clés possibles est souvent infini... mais la mémoire est finie :-).

Clés infinies + mémoire finie = Problème de collisions !

Solution :

- **Limiter** le nombre de hashcodes possibles.



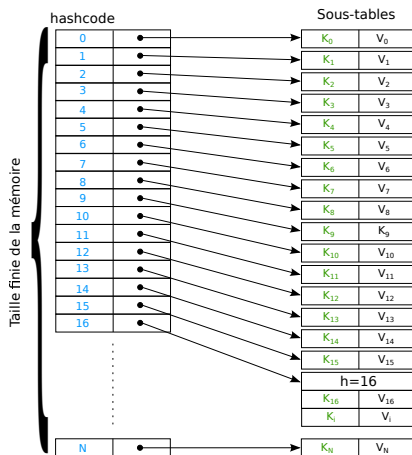
Limitations matérielles

En réalité, le nombre de clés possibles est souvent infini... mais la mémoire est finie :-).

Clés infinies + mémoire finie = Problème de collisions !

Solution :

- **Limiter** le nombre de hashcodes possibles.
- Utiliser une **sous-table** pour les éléments de **même hashcode**.



À vous de créer l'exemple !

sous-tables

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

table d'index

hashcode

0	•
1	•
2	•
3	•
4	•
5	•
6	•
7	•
8	•
9	•

À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1) $k = \text{"tilleul"}$

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

1) $k = \text{"tilleul"}$

2) Somme = $20+9+12+12$
 $+5+21+12 = 91$

À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

1) $k = \text{"tilleul"}$

2) Somme = $20+9+12+12$
 $+5+21+12 = 91$

3) Somme = $91 + 7 = 98$

À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

1) $k = \text{"tilleul"}$

2) Somme = $20+9+12+12$
 $+5+21+12 = 91$

3) Somme = $91 + 7 = 98$

4) Hashcode = $98 \% 10 = 8$

À vous de créer l'exemple !

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

1) $k = \text{"tilleul"}$

2) Somme = $20+9+12+12$
 $+5+21+12 = 91$

3) Somme = $91 + 7 = 98$

4) Hashcode = $98 \% 10 = 8$

H("tilleul") = 8

À vous de créer l'exemple !

sous-tables

Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

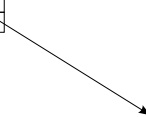
H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**

table d'index

hashcode

0	•
1	•
2	•
3	•
4	•
5	•
6	•
7	•
8	•
9	•



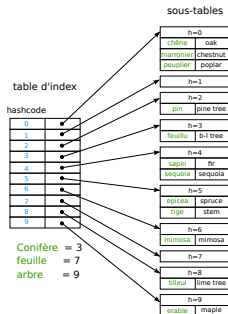
h=8	
tilleul	lime tree

À vous de créer l'exemple !

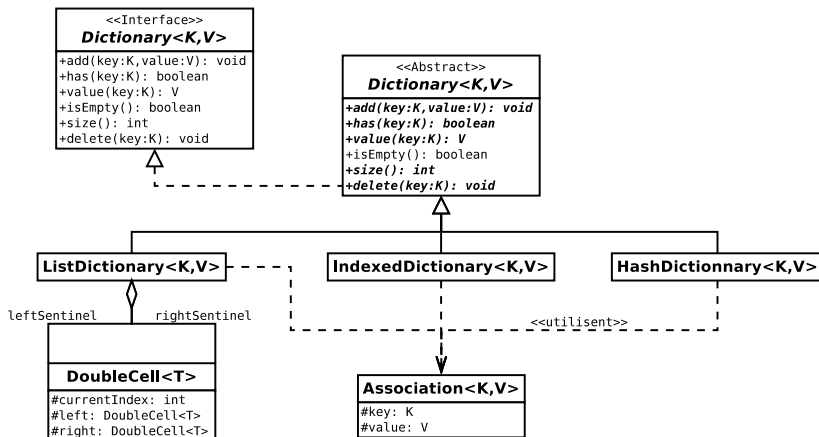
Avec l'ensemble des couples de l'exercice 3.C, complétez le schéma en considérant la fonction de hachage suivante :

H =

1. Attribuer aux lettres les valeurs 1 à 26
2. Sommer les valeurs des lettres de la clé **K**
3. Ajouter au nombre obtenu le nombre de lettres de la clé **K**
4. Calculer le dernier nombre modulo 10 → **hashcode**



Rappel : Diagramme de classes



Exercice 3.F : Énoncé

Écrivez l'implantation Java de la classe `HashDictionary<K,V>` avec représentation par `hashCode`.

- Les sous-tables sont implémentées par des `ArrayList`
- Vous pouvez exploiter la fonction `int hashCode()` de la classe `Object`.

Exercice 3.F : Solution

```
1  public class HashDictionary<K,E>
2      extends AbstractDictionary<K,V>
3  {
4      // Attributs de la classe
5      // -> À compléter en premier */
6
7      public HashDictionary() {...}
8      public HashDictionary( int nbHashCodes ) {...}
9      public int size() { ... }
10     // Retourne le hashcode de la clé key
11     public int hashCode( K key ) { ... }
12     public void add( K key, V value ) { ... }
13     public void delete( K key ) { ... }
14     public V value( K key ) { ... }
15     public bool has( K key ) { ... }
16     // Affiche le contenu de la table sur la
17     // sortie standard
18     public void afficher() { ... }
19 }
```

Exercice 3.F : Solution - Constructeurs

```
1 public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2 {
3     // Attributs de la classe
4     protected int size;
5     protected int nbSousTables;
6     protected Object[] tab;
7
8     public HashDictionary() {
9         // À compléter
10    }
11
12    public HashDictionary( int nbHashCodes ) {
13        // À compléter
14    }
15
16    ...
17 }
```

Exercice 3.F : Solution - Constructeurs

```
1  public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6
7      public HashDictionary() {
8          this(10);
9      }
10     public HashDictionary( int nbHashCodes ) {
11         nbSousTables = nbHashCodes;
12         tab = new Object[nbHashCodes];
13         for ( int i=0 ; i<nbHashCodes ; i++ ) {
14             tab[i] = new ArrayList<Association<K,V>>();
15         }
16         size = 0;
17     }
18     public int size() { // À compléter }
19     ...
20 }
```

Exercice 3.F : Solution - size

```
1  public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6
7      ...
8
9      public int size() {
10         return size;
11     }
12
13     public int hashCode( K key ) {
14         // À compléter
15     }
16
17     ...
18
19 }
```

Exercice 3.F : Solution - hashCode

```
1  public class HashDictionary<K,E> extends AbstractDictionary
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6
7      ...
8
9      public int hashCode( K key ) {
10         return Math.abs(key.hashCode()) % nbSousTables;
11     }
12
13     public void add( K key, V value ) {
14         // À compléter
15     }
16
17     ...
18 }
```


Exercice 3.F : Solution - add

```
1  public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6      ...
7      public void add( K key, V value ) {
8          int h = hashCode(key);
9          ArrayList<Association<K,V>> sousTable =
10              (ArrayList<Association<K,V>>) tab[h];
11          sousTable.add( new Association<K,V>(key,value) );
12          size++;
13      }
14
15      public void delete( K key ) {
16          // À compléter
17      }
18      ...
19  }
```

Exercice 3.F : Solution - delete

```
1 public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2 {
3     protected int size;
4     protected int nbSousTables;
5     protected Object[] tab;
6     ...
7     public void delete( K key ) {
8         int h = hashCode(key);
9         ArrayList<Association<K,V>> sousTable =
10             (ArrayList<Association<K,V>>) tab[h];
11         for ( int i=sousTable.size()-1 ; i>=0 ; i-- ) {
12             Association<K,V> asso = sousTable.get(i);
13             if ( asso.getKey().equals(key) ) {
14                 sousTable.remove(i);
15                 size--;
16             }
17         }
18     }
19     public V value( K key ) { // À compléter }
20     ...
21 }
```

Exercice 3.F : Solution - value

```
1  public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6      ...
7      public V value( K key ) {
8          int h = hashCode(key);
9          ArrayList<Association<K,V>> sousTable =
10              (ArrayList<Association<K,V>>) tab[h];
11          for ( int i=0 ; i<sousTable.size() ; i++ ) {
12              Association<K,V> asso = sousTable.get(i);
13              if ( asso.getKey().equals(key) ) {
14                  return asso.getValue();
15              }
16          }
17          return null;
18      }
19      public bool has( K key ) { // À compléter }
20      ...
21  }
```

Exercice 3.F : Solution - has

```
1  public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2  {
3      protected int size;
4      protected int nbSousTables;
5      protected Object[] tab;
6
7      ...
8
9      public bool has( K key ) {
10         return value(key) != null;
11     }
12
13     public void afficher() {
14         // À compléter
15     }
16
17     ...
18
19 }
```

Exercice 3.F : Solution - afficher

```
1 public class HashDictionary<K,E> extends AbstractDictionary<K,E>
2 {
3     protected int size;
4     protected int nbSousTables;
5     protected Object[] tab;
6     ...
7     public void afficher() {
8         for ( int i=0 ; i<nbSousTables ; i++ ) {
9             System.out.println("Sous-table h=" + i );
10             ArrayList<Association<K,V>> sousTable =
11                 (ArrayList<Association<K,V>>) tab[h];
12             for ( int j=0 ; j<sousTable.size() ; j++ ) {
13                 Association<K,V> asso = sousTable.get(i);
14                 System.out.println("      "
15                                     + asso.toString());
16             }
17         }
18     }
19     // The end !!
```