

« Map non trié »

« Map trié »

« Ensemble non trié »

« Ensemble trié »

Structures non linéaires

L'ordre des ajouts n'a pas d'importance

Map : clé - valeur

Ensemble : élément : clé - null

Unicité des clés (éléments)

**interface EnsembleTrie<E> extends
Iterator<E>:**

`int taille()`

`boolean estVide()`



`boolean contient(E e)`

`boolean ajouter(E e)`

`boolean enlever(E e)`

ET JAVA?


Ensemble	HashSet
EnsembleTrie	TreeSet

 `HashSet` |  `TreeSet` |

`boolean`

`add(E e)`

Adds the specified element to this set if it is not already present.

 `HashSet` |

`Iterator<E>`

`iterator()`

Returns an iterator over the elements in this set.

 `TreeSet` |

`Iterator<E>`

`iterator()`

Returns an iterator over the elements in this set in ascending order.

ET JAVA?

DicoNonTrie	HashMap
Dico	TreeMap

ABR : coût de l'implémentation :

`boolean estVide()`

`int taille()`

`void insere(Comparable element)` $O(\log N)$

`boolean contient(Comparable element)` $O(\log N)$

`void supprime(Comparable element)` $O(\log N)$

Si l'arbre est équilibré :

Implémentation:

Comme l'arbre binaire de recherche (ABR) n'est pas nécessairement équilibré, on va plutôt utiliser une variante de celui-ci :

- Arbre bicolore (\rightarrow B-arbre binaire symétrique)
- AVL (\rightarrow ABR automatiquement équilibré)
- B-arbre (\rightarrow arbre de recherche équilibré)
- ...

ET JAVA?

	HashSet	TreeSet
<code>int size()</code>	$O(1)$	$O(1)$
<code>boolean isEmpty()</code>	$O(1)$	$O(1)$
<code>boolean contains(Object o)</code>	$O(1)$	$O(\log N)$
<code>boolean add(E e)</code>	$O(1)$	$O(\log N)$
<code>boolean remove(Object o)</code>	$O(1)$	$O(\log N)$

ET JAVA?

	HashSet	TreeSet
<code>E first()</code>	/	
<code>E pollFirst()</code>	/	
<code>E floor(E)</code>	/	
...		

Les relations d'ordre

TreeSet, TreeMap et beaucoup d'autres structures de données supposent que leurs éléments peuvent être comparés par une relation d'ordre.

La classe des éléments doit implémenter l'interface Comparable

Interface Comparable

```
Interface Comparable<E> {  
    int compareTo(E e);  
}
```

Si on veut un autre critère de comparaison sachant que `compareTo()` doit être cohérent avec `equals()`, comme on ne peut l'implémenter une deuxième fois → Interface `Comparator`.

Interface Comparator

```
Interface Comparator<E> {  
    public int compare(E e1, E e2);  
    public boolean equals(E e);  
}
```

TreeSet

```
TreeSet()
```

```
TreeSet(Comparator <? Super E> comparator)
```