**Algorithmique / Travaux pratiques**

S06 – Types abstraits « Ensemble » et « Dictionnaire »





25.03.2015

Adriano De Almeida Silva – T-1f

Alex Travasso – T-1f

1. **Explication de la méthode :**

Cette méthode va retourner un ensemble de « Short » contenant toutes les valeurs plus petites que x de l’ensemble de « Short » s.

En effet, elle va parcourir l’ensemble « s » à la recherche des éléments plus petits que « x ».

1. **Implémenter le type abstrait "dictionnaire" spécifié au cours, à l'aide de tableau. Tester.**

**package** s06;

**public** **class** ShortToStringMap {

**private** **int** size;

**private** **int** tabKey[];

**private** String tabValue[];

**private** **int** getPosOfKey(**short** key) {

**for** (**int** i = 0; i < size; i++) {

**if** (tabKey[i] == key) {

**return** i;

}

}

**return** -1;

}

**public** **boolean** containsKey(**short** k) {

**if** (getPosOfKey(k) != -1)

**return** **true**;

**return** **false**;

}

// Public methods

**public** ShortToStringMap() {

tabKey = **new** **int**[2];

tabValue = **new** String[2];

size = 0;

}

// adds an entry in the map, or updates the image

**public** **void** put(**short** key, String img) {

**if** (containsKey(key)) {

**int** x = getPosOfKey(key);

tabValue[x] = img;

} **else** **if** (size == tabKey.length) {

**int** newTabK[] = **new** **int**[size \* 2];

String newTabV[] = **new** String[size \* 2];

**for** (**int** i = 0; i < size; i++) {

newTabV[i] = tabValue[i];

newTabK[i] = tabKey[i];

}

newTabK[size] = key;

newTabV[size] = img;

tabKey = newTabK;

tabValue = newTabV;

size++;

} **else** {

tabKey[size] = key;

tabValue[size] = img;

size++;

}

}

// returns null if !containsKey(key)

**public** String get(**short** key) {

**if** (!containsKey(key)) {

**return** "null";

} **else** {

**int** x = getPosOfKey(key);

**return** tabValue[x];

}

}

**public** **void** remove(**short** key) {

**if** (containsKey(key)) {

**int** pos = getPosOfKey(key);

tabKey[pos] = tabKey[size - 1];

tabValue[pos] = tabValue[size - 1];

tabKey[pos] = tabKey[size - 1];

tabValue[pos] = tabValue[size - 1];

size--;

}

}

**public** **boolean** isEmpty() {

**return** size() == 0;

}

**public** **int** size() {

**return** size;

}

// a.union(b) : a becomes "a union b"

// images are those in b whenever possible

**public** **void** union(ShortToStringMap m) {

**for** (**int** i = 0; i < m.size; i++) {

**this**.put((**short**) m.tabKey[i], m.tabValue[i]);

}

}

// a.intersection(b) : "a becomes a intersection b"

// images are those in b

**public** **void** intersection(ShortToStringMap s) {

**for** (**int** i = 0; i < size; i++) {

**if** (!s.containsKey((**short**) getPosOfKey((**short**) tabKey[i]))) {

remove((**short**) getPosOfKey((**short**) tabKey[i]));

i--;

} **else** {

put((**short**) s.tabKey[i], s.tabValue[i]);

}

}

}

// a.toString() returns all elements in

// a string like: {3:"abc",9:"xy",-5:"jk"}

**public** String toString() {

String resultat = "";

**for** (**int** i = 0; i < size; i++) {

resultat += tabKey[i] + "=" + tabValue[i] + ",";

}

**return** "{" + resultat + "}";

}

}

Nous avons constaté une erreur pour l’intersection que nous n’avons pas réussi à résoudre.



1. **On veut créer un tableau trié de m entiers entre 0 et n-1 (m<=n), tirés au hasard, sans répétition. Implémenter et tester la méthode suivante, en utilisant un ensemble**

public static short[] randomTable(short m, short n) {

short[] result = null;

int i = 0;

SetOfShorts ensemble = new SetOfShorts();

while (ensemble.size() < m) {

ensemble.add((short) *r*.nextInt(n));

}

SetOfShortsItr itr = new SetOfShortsItr(ensemble);

result = new short[m];

while (itr.hasMoreElements()) {

result[i] = itr.nextElement();

i++;

}

Arrays.*sort*(result);

return result;

}