TD-TP N°10– M2103

Semaine 17

Objectifs du TD-TP

Implémenter de algorithmes de recherche et de tri
Lire et écrire des fichiers de texte

Retour sur les classes Pays et Population

Au TD/TP précédent nous avons appris à manipuler des structures de données de type HashMap pour représenter des informations sur des pays.

Maintenant nous allons ajouter des fonctionnalités à la classe Population.

Exercice 1 : Calcul de la densité de population des pays

- Ajouter une méthode à la classe Population qui calcule la densité pour chaque pays. Vous pouvez pour cela ajouter un HashMap pour stocker cette nouvelle information.
- Ajouter une méthode qui retourne le *pays* ayant la plus forte densité de population

Exercice 2 : Tri des noms de pays

En utilisant des HashMap on a perdu l'ordre alphabétique des Pays.

Nous souhaitons trier notre fichier de densité par ordre alphabétique des noms des Pays.

La classe String implémente l'interface **`Comparable`** et possède donc une méthode :

```
public int compareTo(String anotherString)
```

qui retourne
– la valeur 0 si l'argument **`anotherString`** est égale à cette chaîne ;
– une valeur négative si la chaîne est lexicographiquement plus petite que l'argument **`anotherString`**
- et une valeur strictement positive si la chaîne est lexicographiquement plus grande que l'argument **`anotherString`**

Il existe aussi une méthode qui ne tient pas compte des majuscule/minuscule :
`public int compareToIgnoreCase(String str)` : compare deux chaînes en ignorant la casse.

A faire en TP :

- Réaliser le tri de manière similaire du dernier TP, c'est-à-dire en utilisant les classes de tri se trouvant dans le package *tri*.
- Sauvegarder dans un fichier « worlddensity.txt » les informations de ce nouveau HashMap. On ajoutera si nécessaire une méthode pour écrire le fichier dans la classe RWFile.

Une autre solution consiste à explorer le framework des collections Java et à s'intéresser aux collections triées :

`public interface SortedMap<K,V> extends Map<K,V> :` A map that further guarantees that it will be in ascending key order, sorted according to the *natural ordering* of its keys.

All keys inserted into a sorted map must implement the Comparable interface.

Exercice 3 : Recherche dichotomique dans une liste triée (vu en 1103)

La recherche dichotomique est à réaliser sur une liste triée.

Rappel du principe (cours 1103) :

On suppose le tableau leTab trié par ordre croissant.

- Un indice m tel que d <= m < = f, avec d, f Є*0 .. n*
  - –si valeur <= leTab[m].getCle(),
    - alors l'élément recherché (la clé), s'il est dans le tableau, est nécessairement dans l'intervalle [d .. m],
  - sinon
    - il ne peut être que dans l'intervalle [m+1 .. f].
- En répétant la subdivision de l'intervalle plusieurs fois, on «encadre» la valeur dans un intervalle de plus en plus petit, jusqu'à ce que cet intervalle ne contienne plus qu'une valeur.
- À ce point, soit la leTab[m] est égale à valeur, et on a trouvé, sinon elle n'est pas dans le tableau.

L'algorithme de recherche tel qu'il est dans le cours 1103

```
int rechercheDicho (int [ ] leTab, int nb, int aRech ) {
   // variables locales
   // indD = indice de début du sous-tableau
   // indF = indice de fin du sous-tableau
   // indM
   //indice milieu entre indD et indF
   int indD, indF, indM, ret;
  // initialisations
   indD = 0;
   indF = nb – 1;
   // boucle
    while ( indD != indF ) {
           indM = ( indD + indF ) / 2; // division entière !
          if ( aRech > leTab[indM] ) {
            indD = indM + 1;
          }
          else indF = indM ;
    }

 // terminaison : indD == indF forcément
    // MAIS leTab [indD] par forcément = aRech !!
        if ( aRech == leTab [indD] ) ret = indD;
        else ret = -1;
        return ret
}
```

Cet algorithme consiste donc à couper en deux la liste et à regarder si l'élément que l'on recherche est dans la première ou la deuxième moitié.
On continue la même recherche sur la moitié concernée et on s'arrête si on a trouvé l'élément recherché ou si on a atteint la fin de la liste.

A faire : implémenter la recherche dichotomique en Java en l'appliquant à la recherche d'un nom de pays.

Exercice Bonus

Reprenez votre cours de M1103 et implémenter un tri rapide. Vous comparerez ensuite le temps mis par un

tri rapide en comparaison à un tri simple.

A faire en TP:

Après avoir implémenté ce qui est demandé dans les exercices 1,2 et 3, écrire une classe de scénario qui propose une interaction pour rechercher un pays dans la liste des pays. Il faut

- trier la liste des Pays par ordre alphabétique
- avoir un algorithme de recherche qui fonctionne pour rechercher un pays
- faire une petite boucle de lecture au clavier qui demande à l'utilisateur de taper au clavier le nom d'un pays et qui retourne les informations sur le pays (population, surface, voire densité).