Introduction

L'objectif de ce TP est d'implémenter un algorithme de tri externe en Java comme vu en cours. Cet algorithme prendra comme premier argument un fichier CSV contenant un en-tête contenant les noms des colonnes sur la première ligne, le reste des lignes contient des n-uplet. Le deuxième argument sera la liste des noms de colonne à utiliser pour le tri, exactement comme l'argument qui suit `ORDER BY` en SQL. La classe `TriExterne`, que vous allez implementer, utilise une mémoire cache de taille `M` pour appliquer un algorithme de fusion en stockant des fragments du fichier à trier sur le disque.

Vu que le projet contient plusieurs fichiers, on compile avec la commande :

```
javac *.java
```

Puis, on éxécute avec une commande de la forme :

```
java TriExterne fichier.csv col1;col2;...
```

On peut faire bien sûr faire le tout avec une commande :

```
javac *.java && java TriExterne fichier.csv col1;col2;...
```

Questions

Introduction à la classe `TriExterne`

L'objectif de la classe `TriExterne` est de trier un fichier CSV qui ne tient pas forcément en mémoire à l'aide d'une quantité de mémoire fixé par le paramètre `M`. Le paramètre `cache` de la classe permet de stocker `M` n-uplets en mémoire.

Les questions suivantes introduisent le code, il y a pas besoin de répondre aux questions dans un document, des commentaires dans le code sont suffisants.

Question 1. En lisant la documentation de BufferedReader, donner l'utiliser du paramètre `fichierInit` et son utilisation mémoire.

Question 2. Dans le constructeur de `TriExterne` que représente le tableau `indice` ? Commentez le code.

Implémentation du comparateur

Le comparateur permet de choisir un ordre sur des n-uplet en fonction d'une sous liste de leurs colonnes.

Question 3. Implémenter la méthode `int compare(String[] t1, String[] t2)` de la classe `Comparateur`, qui renvoit un entier positif si `t1` est strictement plus grand que `t2`, 0, s'ils sont égaux et négatif, sinon. On utilise l'ordre lexicographique sur les chaînes de caractères étendues

aux listes de chaînes de caractères. Les colonnes qui sont utilisées pour faire la comparaison sont celles dont les indices se trouvent dans le paramètre `indices`.

Question 4. Dans la classe `TriExterne`, créez une méthode pour tester le comparateur sur des n-uplet en faisant une comparaison en utilisant une colonne, deux colonnes.

Question 5. Que se passe-t-il si vous comparez les n-uplets (9) et (10) ? Pourquoi ?

Creation des fragments initiaux

Le but de cette étape est de fragmenter le fichier CSV initial en fragments qui peuvent être stockés en mémoire cache. Puis, de trier ces n-uplets, et de les stocker dans un fichier. Ces premiers fragments sont stockés dans des fichiers CSV nommés `fragment_0_i.csv` où i est le numéro du fragment. On crée autant de fragment que nécessaire pour avoir tous les n-uplets.

Question 6. Coder la méthode `trierCache(int taille)`, qui trie les n-uplets du cache entre les indices 0 et `taille-1` inclus, selon le comparateur fourni. **Indice** : Voir la documentation de `Arrays.sort`.

Question 7. Coder la méthode statique `ecrireLigneCSV(FileWriter fw, String[] valeurs)`, qui écrit une ligne dans un fichier CSV ouvert avec le `FileWriter`, en séparant les valeurs par des points-virgules.

Question 8. Coder la méthode `sauvegardeCache(String path, String[] entete, int taille)`, qui sauvegarde le contenu du cache (de l'indice 0 à `taille-1`) dans un fichier CSV de chemin `path`, en écrivant d'abord l'entête.

Question 9. Coder `creerFragmentsInitiaux()`, qui lit le fichier CSV initial, remplit le cache, trie chaque fragment de taille `M`, et sauvegarde chaque fragment dans un fichier correspondant.
Indice : Utiliser la méthode `nomDeFragment` pour créer les noms des fragments, le niveau initial est 0 et on numérote les fragments de 0 à n. La méthode retourne le nombre de n-uplets du fichier initial.
Astuce : Utiliser le code suivant pour lire ligne par ligne le fichier initial.

```
String ligne;
while ((ligne = this.fichierInit.readLine()) != null)
....
```

Question 10. Compléter la méthode `trier` pour qu'elle génère les fragments initiaux. Tester votre code avec le fichier CSV `communes.csv`.

Fusion

Pour rappel, une fois que les fragments initiaux sont créés le tri externe les fusionnent par groupe de M-1 fragments pour former de nouveaux fragments de niveau supérieur. L'algorithme arrête dès qu'un niveau ne contient qu'un seul fragment, c'est à dire le fichier initial trié. La méthode `int fusion(int niveau, int nombre)` effectue la fusion des fragments d'un niveau donnée sachant que `nombre` est le nombre de fragments à ce niveau.

Question 11. Dans un premier temps, on va supposer que le nombre de fragment à fusion du niveau est inférieur ou égal à M-1. On peut alors procéder de la manière suivante :

1. ouvrir tous les fichiers à fusionner en parallèle avec un tableau de `BuffferedReader`
2. charger le premier n-uplet de chaque fichier dans les M-1 premières cases du cache
3. trouver le n-uplet minimal sans créer de nouvelle variable pour stocker un n-uplet (indice la dernière case du cache est disponible)
4. écrire ce n-uplet dans le nouveau fragment

5. répéter jusqu'à épuisement de l'ensemble des fichiers

Indices : Il faut garder l'indice du fichiers qui contient le n-uplets minimal pour mettre à jour le cache. Quand un fichier est vide, on peut mettre son n-uplet à null dans le cache.

Question 12. Compléter à nouveau la méthode `trier` pour qu'elle effectue la fusion des M-1 premiers fragments initiaux. Tester la question précédente avec `communes.csv`.

Question 13. Modifiez votre méthode `fusion` pour les cas où le nombre de fragments à fusionner est supérieur à M. Tester à nouveau en modifiant `trier` pour qu'elle génère l'ensemble des fragments de niveau 1.

Question 14. Complétez la méthode `trier` pour qu'elle effectue le tri externe.

Aller plus loin

Gérer le type des colonnes

Modifiez le code pour que le tri soit effectuer en prenant en compte que certaines colonnes sont numériques et d'autres textuelles. On précise le type des colonnes avec un nouvel argument, par exemple :

```
java TriExterne fichier.csv col1;col2;... NUM;TXT;...
```

Optimisation

Optimisez votre implémentation du tri externe en utilisant une structure de données plus efficace pour le cache, comme un tas.