Algorithmique et structures de données : Mission 4 correction croisée

Groupe 1.2: Ivan Ahad - Jérôme Bertaux - Rodolphe Cambier
Baptiste Degryse - Wojciech Grynczel - Charles Jaquet

14 novembre 2014

Rapport écrit par Rodolphe Cambier, Ivan Ahad et Charles Jaquet.

Introduction

A la suite de la mission précédente où l'objectif était de faire un programme permettant d'accéder au classement d'une revue scientifique, cette mission-ci consiste en l'implémentation d'un dictionnaire ordonné, étant donné qu'on doit pouvoir avoir une liste de ces revues dans l'ordre alphabétique.

Fonctionnement

Notre code de la mission précédente permettait déjà la création d'un dictionnaire ordonné, cette mission-ci reprend le même code en rajoutant une interface graphique ainsi qu'un moyen de retourner le dictionnaire trié.

Pour la partie graphique, les classes *Fenetre* et *EntreePanel* sont rajoutées.

Pour trier le dictionnaire, la classe *EntreeComparable* est rajoutée. Ainsi que la fonction *List<Entree> getSortedList()*, utilisée afin de trier la Map contenant les journeaux.

La classe *Performance* est ajoutée pour de réaliser un test de performance.

Le code fonctionne comme suit:

- La classe **Main** crée un **Dictionnaire**.
- Le **Dictionnaire** lit le fichier, met chaque ligne sous forme clé-valeur dans une **Map** ordonnée, en le récupérant de la mission précédente.
- La classe **Main** crée une **Fenetre**
- La **Fenetre** demande une fieldName et fieldValue (Catégorie à rechercher, et String à rechercher dans cette catégorie) ainsi qu'un champ orderBy (Catégorie selon laquelle trier les résultats)
- La **Fenetre** va print les résultats après avoir utilisé *getSortedList()* sur le dictionnaire.

Tests réalisés

La classe *DictionnaireTest* est utilisée afin de tester le programme.

Plusieurs fonctions testent différentes partie du programme, et les différentes limites de celui-ci.

Questions par rapport au programme

Question 9(Charles) - Performance

Pour analyser les performances du tri, j'ai dû modifier le programme pour lui permettre de faire des dictionnaires avec un nombre d'éléments prédéfini. Ensuite j'ai fait les calculs du temps moyens pour créer une liste triée des éléments. Enfin j'ai écris ces temps dans un fichier qui m'a permet de tracer le graphe (fig. 1) du temps nécéssaire pour créer une liste ordonnée en partant de la treeMap en fonction du nombre d'éléments.

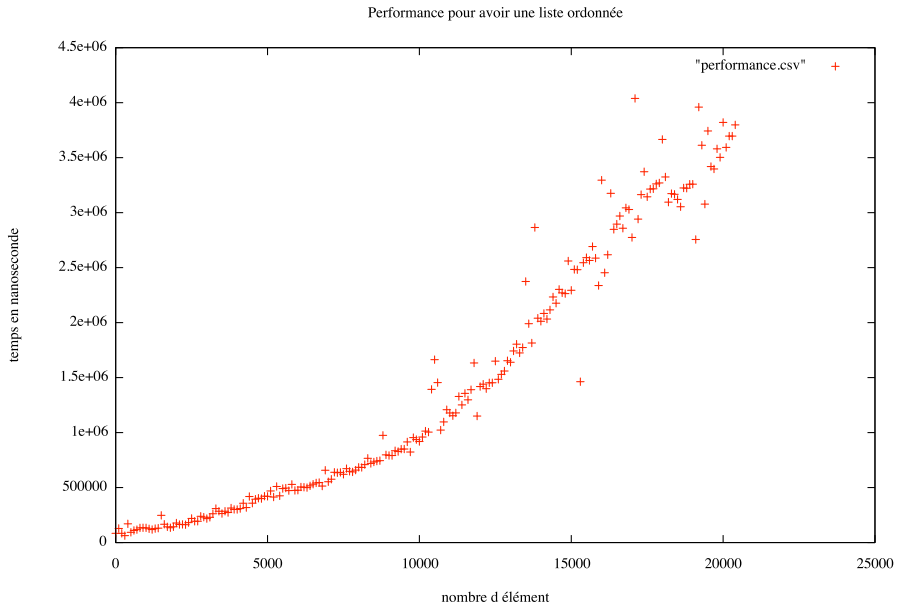

Figure 1: Graphe du temps pour créer une liste ordonnée en partant du TreeMap en fonction du nombre d'éléments

Ensuite, en fonction du nombre d'éléments, nous avons calculé le temps pris pour trouver un élément dans la liste. (Fig. 2)

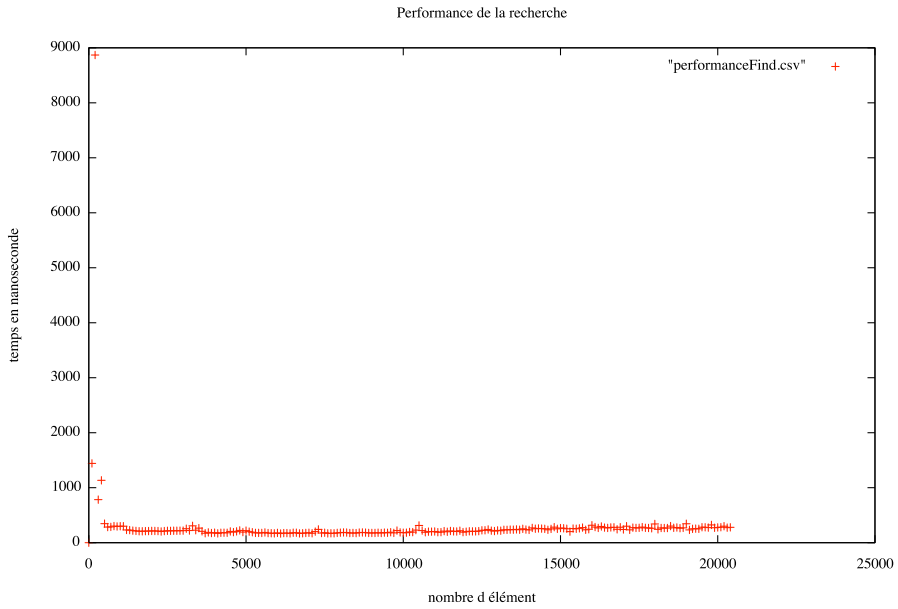

Figure 2: Temps moyen mis pour trouvé un élément

Etonnement, nous avons un temps stable or que nous nous attendions à ce que ça évolue de manière logarithmique

Question 10

Toute la partie de création d'une **Map** à partir du fichier texte a pu être réutilisée.

Il a fallut implémenter un système de tri, et donc rajouter les classes comme décrit dans la partie **Fonctionnement**. Nous avons également choisi d'implémenter une interface graphique.

Le code est plus générique, car il permet de trier dans l'ordre voulu et de choisir le fichier source. La lecture et l'écriture du fichier se fait maintenant dans une classe à part, afin de rendre le fichier plus modulable.

Question 11

Les lignes erronées ne sont simplement pas ajoutées dans la **Map** et n'interfèrent donc pas avec le reste du programme.

Schéma UML

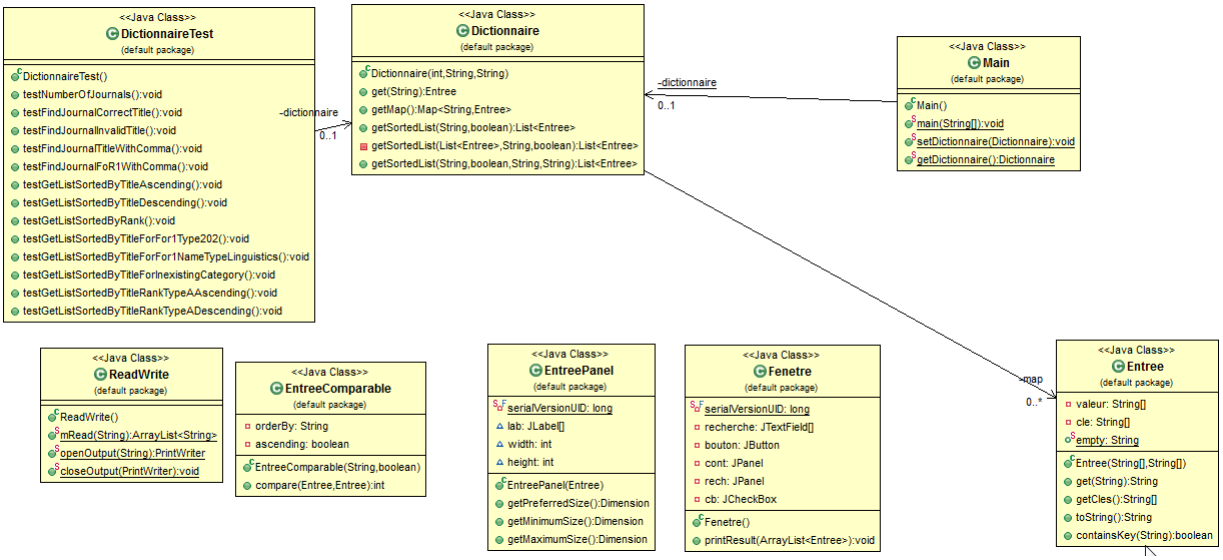

Figure 3: Diagramme UML