Rapport - Projet Java

JSPELL

IR1

Nacer Dergal

Leroux Gwenaël

Sommaire

I / DictionaryCreator

1) L'architecture

Pour la classe DictionaryCreator, nous avons créé une TreeMap avec chaque mot du fichier d'entrée.

La TreeMap a comme clé le mot et en valeur le mot et son nombre d'occurrences.

L'usage de la TreeMap nous apporte deux avantages :

- Nous avons une structure triée par ordre alphabétique des clés.
- La recherche en O(1).

Le programme récupère l'ensemble des mots du fichier. Il les insère tous dans la TreeMap sauf si le mot existait déjà dedans. Dans ce cas, on incrémente son nombre d'occurrences. A la fin on affiche tous les valeurs de la TreeMap sur la sortie standard.

2) Les difficultés rencontrées et leurs solutions

Nous avions rencontré un problème pour obtenir un dictionnaire trié. Nous avions commencé par utiliser une ArrayList<WordDictionary>. Cette implémentation provoquait une lenteur d'exécution car nous recherchions pour chaque mot si il existait déjà dans l'ArrayList.

Nous avons opté pour une TreeMap qui nous permet d'avoir une complexité en O(1) pour rechercher si le mot existe déjà. Avec cette implémentation, nous obtenons un dictionnaire trié.

II / SpellAnnotator

1) L'architecture

La classe SpellAnnotator permet de récupérer les mots tapés sur l'entrée standard, de déterminer quel dictionnaire utilisé parmi ceux passés en arguments du programme, d'afficher le mot sur la sortie standard ou le mot entre balise <spell></spell> avec les suggestions de corrections les plus pertinentes si le mot n'est pas dans le dictionnaire.

Nous avons implémenté trois modules de propositions :

- Le soundex
- La distance de hamming (gauche et droite)
- La distance de Levenshtein

Lors de l'appel au constructeur de SpellAnnotator, il construit une ArrayList de tous les mots qui lui sont passés en argument.

Nous créons ensuite une ArrayList<Dictionary> contenant tous les dictionnaires passés en paramètre du programme. Lors de l'appel au constructeur de Dictionary, il crée une TreeMap de tous les mots du fichier dictionnaire donné en argument. Pour chacun des dictionnaires créés, nous comptons le nombre de mots inconnus et nous choisissons celui qui en possède le moins.

Pour chacun des mots entrés sur l'entrée standard, nous regardons si il est dans notre dictionnaire. La TreeMap du dictionnaire nous assure une recherche en O(1). Lorsque nous arriverons sur un mot inconnu, nous regardons si nous avons déjà récupéré ses suggestions. Dans le cas contraire, nous créons une ArrayList contenant les mots du dictionnaire ayant le même soundex, une avec ceux ayant la distance de Hamming la moins élevée et une troisième pour ceux qui ont la distance de Levenshtein la moins élevée. Nous

concaténons ces trois ArrayList que nous trions par occurence et nous renvoyant une ArrayList contenant les cinq meilleures suggestions.

III / SpellSelector

1) L'architecture

La classe SpellSelector permet d'appliquer une correction à un fichier de texte contenant des erreurs qui ont été repérés par la classe SpellAnnotator. Pour des mots placés entre les balises nous affichons les 3 mots précédents s'ils existent et nous proposons soit de:

- Le remplacer par ses suggestions.
- L'ignorer pour cette fois.
- L'ignorer définitivement pour cette session de correction auquel cas nous l'ajoutons à une HashMap de mot ignorés ce qui nous permet de tester si les mots sont ignorés ou non en O(1).
- De l'ajouter au dictionnaire.

Si le nombre de mots du dictionnaire est changé nous le réécrivons.

IV/ Dictionary

1) L'architecture

Un dictionnaire contient son nom qui est aussi son chemin, sa langue, son nombre de mots qui nous permet de savoir en O(1) sa taille et une TreeMap des mots et de leurs fréquences qui nous permet de savoir quelles mots sont contenu dedans et aussi faire des ajouts en O(1).

V/ Word

1) L'architecture

La classe Word contient comme champs un String.

VI/ WordDictionnary

La classe WordDictionnary est une classe de mots comprit dans le dictionnaire, elle hérite de Word et possède un champs en plus qui est la fréquence. Cela nous permet de garder un mot et sa fréquence d'apparition dans un même objet.