Manuel développeur

Architecture:

Dans notre programme, quatre types d'Objet existent.

L'Objet Dictionary

Il contient l'ensemble des méthodes permettant de construire, chercher et ajouter un mot dans un dictionnaire mais aussi récupérer les mots similaires.
Pour faire tout cela, un Dictionary possède 5 champs:

- Une HashMap<String, Integer> qui regroupe l'ensemble des mots ainsi que leur nombre d'apparitions dans le texte lu pour créer le dictionnaire
- Un String pour garder son nom
- Un File pour ajouter un mot dans le fichier
- Un Locale pour le module Soundex
- Un ModuleAggregator est un Objet permettant de faire des recherches. (voir ci-dessus)

Améliorations possibles:

- Faire un parcours de corpus qui mettrait à jour les occurrences des mots car pour le moment, nous lisons des fichiers dictionnaires dans lesquels les mots apparaissent seulement une fois (mais si le mot apparait plusieurs fois, il met bien à jour les occurrences)

L'Objet ModuleAggregator

Il contient deux méthodes: une pour rechercher des mots similaires à un mot dans en argument à l'aide de Modules et une autre qui ajoute un mot aux Modules.
Pour fonctionner, celui-ci construit une List de Module qu'il gardera en mémoire et mettra à jour quand cela lui est demandé.
Dans notre implémentation, le module de Damerau-Levenshtein est prioritaire sur celui de Hamming qui est supérieur au module Soundex.

Améliorations possibles:

- Pour le moment, nous tronquons la concaténation des résultats des Modules alors qu'il serait plus intelligent de regarder si un mot apparait dans plusieurs modules et donc le mettre en priorité dans les résultats.
- De même, on pourrait utiliser le nombre d'appariation pour favoriser certains résultats.

L'Interface Module et ses implémentations

Cette interface définit deux méthodes: getNearestSiblings et updateModule.

La première sert à récupérer les mots qui sont les plus proches du mot donné en argument, la deuxième à mettre à jour le module avec le mot donné en argument.
updateModule est définit par défaut comme ne faisant rien.

Damerau-Levenshtein

Le module DamerauLevenshtein implémente l'algorithme du même nom ; il se sert du Dictionary qui l'a créé, qu'il parcourt afin de regarder quels sont les mots les plus proches selon l'algorithme de calcul de distance. Ce module renvoie les trois mots les proches du mot donné en argument à getNearestSiblings. Pour l'updateModule, comme le module s'appuie sur le dictionnaire, il n'a pas besoin de se mettre lui-même à jour.

Hamming

Le module Hamming implémente l'algorithme du même nom ; il a une structure de données qui lui est propre. Celui-ci possède deux champs qui lui reservé: une String conservant l'origine de la structure et une HashMap<Integer, Set<String>> qui est la représentation du module.

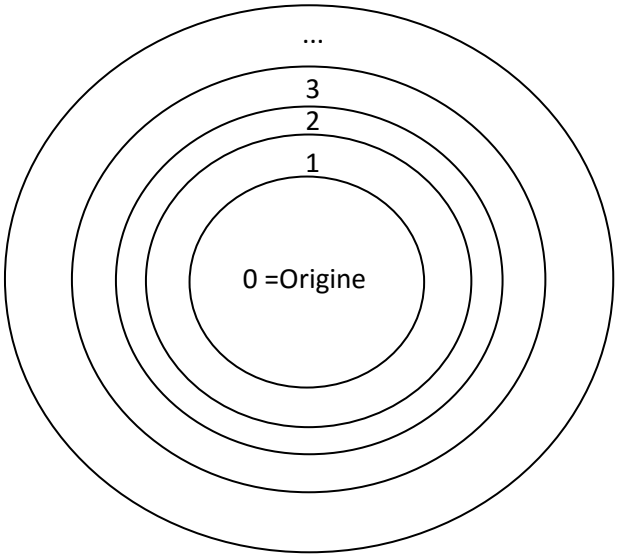

Figure 1 : représentation graphique de la structures de données

Comme le montre la figure ci-dessus, on a regroupé les mots par distance par rapport au mot d'origine.
Ainsi pour faire une recherche, on ne parcourt pas tout le dictionnaire mais l'ensemble de mots dont la distance de Hamming est ±2 celle entre le mot donné en argument et l'origine.

Soundex

Le module

L'Objet Corrector

Il contient toutes les méthodes de découpages de mots, lignes, de demande de mots proches et de lecture/écriture dans un fichier. C'est le coordinateur de la correction. Pour fonctionner, il a besoin de Dictionary.
Lorsque l'on veut corriger un texte:

- il commence par sélectionner le meilleur dictionnaire parmi ceux qu'il possède
- puis il parcourt le fichier en créant un fichier .anot dans lequel tous les mots incorrects sont balisés
- ensuite il parcourt le fichier .anot et demande à chaque fois qu'il voit une balise par quelle action faire à l'utilisateur et écrit dans un fichier .tmp
- Une fois tout le parcours effectué, il supprime le fichier .anot, supprime le fichier d'origine et le remplace par le fichier .tmp.

Nous avons fait le choix de faire deux fichiers temporaires en cas d'erreur dans la correction ou d'arrêt en plein milieu (pour ne pas avoir à ré annoter).

Améliorations possibles:

- faire une option pour arrêter la correction et mettre la fin du fichier tel qu'il était.

Modularité:

Dans notre projet, il est simple d'ajouter un module. Il suffit de créer un module qui implémente l'interface Module et de définir la méthode getNearestSiblings (et updateModule si besoin). Une fois le nouveau Module définit il ne reste plus qu'à l'ajouter à la List dans ModuleAggregator et le tour est joué.