Projet de Java – Foogle

Licence 3 – Informatique

Rémi Forax, Etienne Duris
(forax@univ-mlv.fr, duris@univ-mlv.fr)

Description du Projet

Le programme **Foogle** (foo-google) est un programme qui crée un index à partir d'un fichier et qui permet de faire des requêtes de recherche de mots en utilisant les opérateurs **ET** et **OU** pour trouver l'ensemble des lignes satisfaisant la requête.

Calendrier

Ce projet est à faire par binômes (cela veut dire deux personnes pas trois ni une).
Le projet est à rendre AVANT le **dimanche 19 juin** à 23h59 par par mail sous forme d'une pièce jointe au format ZIP contenant l'ensemble des programmes et documents.
Le mail devra être envoyé avant minuit aux deux adresses données au début de ce document.

Le fichier s'appellera nom1_nom2.zip avec nom1 et nom2 les nom des binômes dans l'ordre alphabétique.

Ligne de Commande du Logiciel

Le logiciel doit se présenter sous la forme d'un jar exécutable.
La ligne de commande permet de créer l'index ou d'effectuer des requêtes sur un index déjà crée

java -jar foogle.jar [index|query] [option spécifique dépendant de index ou query]

index :

java -jar foogle.jar index [-strategy name] [-minlength length]fichier.ext
crée un index à partir d'un **fichier.ext**, le nom du fichier sera **fichier.ext.idx**
-minlength indique le nombre de lettre minimum du mot pour qu'il appartienne à l'index.
-strategy permet de spécifier une stratégie de contexte par son nom.
Le contexte correspond aux mots avant et après un mot recherché.
(par défaut: none)

La stratégie "none" indique que le contexte (la ligne ou une partie de la ligne n'est pas stockée dans l'index.
La stratégie "near" indique que le contexte correspond au mot précédent ainsi qu'au mot suivant le mot courant.

Exemple de résulta avec un contexte "near", avec le fichier :

```
hello toto how are you ?
well, i'm fine, well Dee fine.
```

Pour la requête toto & how (les contextes sont agglomérés):
line 1: hello toto how are

Pour la requête fine (le mots fine est présent plusieurs fois):
line 2: well fine well ... Dee fine

De façon optionnel vous pouvez ajouter un contexte intelligent.

query :

java -jar foogle.jar query [-n count] fichier.ext.idx expr
effectue une requête **expr** sur l'index **fichier.ext.idx** et affiche l'ensemble des lignes satisfaisant la requête.
-n permet de spécifier le nombre de résultat voulu.

Les requêtes sur l'index correspond à un arbre
au format préfixé,
exemple "& toto | titi @t?t?" est équivalent à l'arbre

L'opérateur **ET** ('&') reporte la ligne uniquement si les deux mots appartiennent à la ligne.
L'opérateur **OU** ('|') reporte la ligne si un des deux mots appartient à la ligne.
L'opérateur @ permet d'indiquer une expression régulière (au format PERL) pour rechercher les mots.

De façon optionnel, il est possible dans le cas des **OU** lorsque l'on fixe le nombre de résultat voulu de ne pas calculer l'ensemble de tous les résultats possibles. De plus, lors de requête concernant beaucoup de résultat possible, il est possible de manipuler des itérateurs plutôt que des listes pour éviter d'avoir de grosses structures de données intermédiaires.

Architecture du Logiciel

Pour vous aidez et nous permettre de tester automatiquement votre logiciel,
un ensemble d'interface est imposée, vous ne devez en aucun cas toucher leur code.

Exemples d'utilisation des interfaces :
Création d'un index :

```
FoogleFacade foogle=new FoogleFacadeImpl();
ContextStrategy contextStrategy=
  foogle.getContextStrategyMap().get("none");

Index index=foogle.createIndex(
  new File("bbe.txt"),contextStrategy);

foogle.saveIndex(index,new File("bbe.idx"));
```

Recherche dans un index :

```
Index index2=foogle.loadIndex(new File("bbe.idx"));
System.out.println("indexed filename:"+index2.getFilename());

QueryFactory factory2=foogle.createQueryFactory(index2);
Query query2=factory2.and(factory2.query("God"),
  factory2.query("solid"));

ContextStrategy contextStrategy2=index2.getContextStrategy();
for(Result result2:query2.execute(10)) {
  System.out.println("line:"+result2.getLine());
  System.out.println("ctx:"+contextStrategy2.getContext(result2));
}
```

Détail du Rendu

Voici le nom des répertoires et fichiers qui doivent être contenus dans l'archive ZIP.

1. un fichier **readme.txt** indiquant comment compiler et exécuter le programme, où se trouve la doc, *etc.*
2. un fichier Ant **build.xml** permettant de compiler les sources du programme, et de créer le jar exécutable **foogle.jar**.
3. un répertoire **src** contenant l'ensemble des sources (.java) sous forme de plusieurs paquetages. Les interfaces et classes contenues dans **fr.umlv.foogle** ne doivent pas être modifiées.
4. un répertoire **classes** contenant l'ensemble des classes (.class) correspondant aux sources sous forme de plusieurs paquetages.
5. un répertoire **bin** contenant deux fichiers, **foogle.sh**, script shell démarrant le logiciel sous Linux (en fait Unix) et **foogle.bat** démarrant le logiciel sous Windows.
6. un répertoire **docs** contenant deux documents au format PDF :
    1. La documentation utilisateur **user.pdf** contenant en plus des informations classiques (comment compiler, exécuter, etc.) une description de l'application, et comment l'utiliser
    2. La documentation développeur **dev.pdf** contenant :
       Une description détaillé et au niveau de chaque classe que vous avez implantée.
       Evitez les copier/coller de la javadoc, essayer plutôt de décrires les objets (par

leur responsabilité), les structures de données et les algorithmes dans un ordre logique.
Une liste des bugs connus (s'il y en a) détaillant le scénario permettant de générer le bug ainsi que la raison pour laquelle celui-ci se produit.
En plus des deux documents, le répertoire **docs** devra contenir un sous-répertoire **api** contenant la documentation complète du logiciel au format javadoc.