UNIVERSITÉ D'AVIGNON
ET DES PAYS DE VAUCLUSE

M2 ILSEN – 2019/20

UE Ingénierie du document et de l'information

UCE Indexation & recherche

Vincent Labatut

Préparation | Examen de travaux pratiques

L'examen de TP utilisera un certain nombre de méthodes déjà écrites à l'occasion d'examens ou de TP précédents. Le but de ce document est d'expliquer lesquelles, et comment les inclure dans votre code source, afin de le préparer convenablement à l'examen. L'idée est de gagner du temps durant l'examen lui-même, afin de pouvoir vous concentrer sur les points importants (en termes de méthodes d'indexation) ciblées par l'examen. Vous pouvez considérer que l'examen de TP est divisé en deux parties, et que ce document constitue la première de ces parties.

1 Préparation

Comme indiqué à l'avance, on se basera sur l'implémentation réalisée auparavant lors de la série de TP. Faites une copie de votre projet Eclipse, à laquelle vous devez donner un nom de la forme `XxxxYyyy`, où `Xxxx` est votre nom de famille et `Yyyy` votre prénom. Par exemple, l'étudiant *Ahmet Potier* appellera son projet `PotierAhmet`. Attention à bien respecter cette consigne de nommage. On travaillera ensuite exclusivement dans cette copie-là du projet. L'archive que vous rendez en fin de TP devra porter le même nom que votre projet. Notez que des classes-squelettes (i.e. à compléter) seront fournies en début de séance.

L'examen de TP portera sur la recherche approximative. Par conséquent, il est nécessaire de modifier certaines méthodes et classes qui interviennent à la fois lors de la construction de l'index et lors du traitement des requêtes. Ces modifications ont été traitées lors de l'examen de TP de l'année 2016/17 (plus précisément, sa Section 3), dont les corrections sont disponibles sur e-uapv. Vous devez bien entendu tirer parti de ces corrections pour appliquer les modifications demandées dans la suite de ce document.

2 Tokénisation & Normalisation

La recherche approchée à base de jokers repose sur l'utilisation de caractères spéciaux dans les requêtes, pour permettre à l'utilisateur d'indiquer explicitement les parties inconnues. Ces caractères sont traités différemment lors de la construction de l'index et lors du traitement des requêtes : dans le premier cas, ils doivent être considérés comme des séparateurs (on part du principe que les documents ne contiennent pas de jokers), alors qu'ils doivent être traités comme des jokers dans le second cas. Le but de cette partie est de mettre en place cette distinction. On se concentrera uniquement sur le joker `*` dans le cadre de cet examen, qui pourra se trouver n'importe où dans un mot (début, intérieur, fin).

Exercice 1

Le tokénisateur précédemment défini en TP pose problème, car il sépare les mots en fonction de tous les caractères qui ne sont ni des chiffres ni des lettres. C'est très bien pour indexer le corpus, mais pas pour traiter les requêtes. En effet, si on l'applique en l'état à une requête telle que `arb* plant`, le joker `*`, qui n'est ni un chiffre ni une lettre, va simplement disparaître et on obtiendra les tokens `arb` et `plant`, dont le premier est erroné.

Pour éviter cela, dans la classe `Tokenizer`, modifiez la méthode `tokenizeString` de manière à ce qu'elle prenne un second paramètre de type booléen appelé `removeJokers`. Si ce paramètre est vrai, la méthode fonctionne normalement ; et s'il est faux, elle traite les jokers comme les chiffres et lettres. Il suffit pour cela de rajouter `*` dans la classe regex (`[...]`) utilisée lors de l'invocation de `split` dans `tokenizeString`.

Exemples :

- `tokenizeString("La fleur",true)` renvoie `"La"` et `"fleur"` ;
- `tokenizeString("La fleur",false)` renvoie `"La"` et `"fleur"` ;
- `tokenizeString("La* fleur",true)` renvoie `"La"` et `"fleur"` ;
- `tokenizeString("La* fleur",false)` renvoie `"La*"` et `"fleur"`.

Adaptez ensuite la méthode `tokenizeDocument` à ce changement : `tokenizeString` doit désormais y être invoquée avec la valeur `true`, de manière à garder l'*ancien* comportement de `tokenizeString` lors de la construction de l'index.

Allez enfin dans la classe `AndQueryEngine`, et modifiez la méthode `splitAndQuery` de manière à ce qu'elle invoque `tokenizeString` avec la valeur `false`, au contraire[1], de manière à basculer sur le *nouveau* comportement de `tokenizeString` lors du traitement des requêtes.

> Aidez-vous de la correction de l'Exercice 5 de l'examen de TP 2016/17.

3 Recherche dans l'index

Il est nécessaire de faire quelques modifications dans les classes relatives au traitement des requêtes, afin d'intégrer la possibilité d'une recherche approchée. Ces modifications seront poursuivies lors de l'examen.

Exercice 2

On n'utilisera pas les lexiques à base de *tableau* ni de *table de hachage* lors de l'examen de TP. Pour éviter toute erreur, supprimez les classes `ArrayIndex` et `HashIndex`, ainsi que les lignes concernant ces classes dans la méthode `Builder.buildIndex` et dans le type énuméré `AbstractIndex.LexiconType`.

Exercice 3

Dans la classe `AbstractIndex`, ajoutez une méthode `public abstract List<IndexEntry> getEntriesStartingWith(String prefix)` qui prend en paramètre une chaîne de caractères `prefix` et renvoie la liste de toutes les entrées dont le terme commence de façon similaire à `prefix`. Écrivez l'implémentation de cette méthode dans `TreeIndex`.

Exemple : pour le préfixe `la`, on pourrait obtenir `lacet`, `lard`, `latéral`, etc.

> Aidez-vous de la correction de l'Exercice 7 de l'examen de TP 2016/17. Attention, il faut faire quelques adaptations au code source proposé, car celui-ci utilise un `TreeSet` pour représenter l'index, alors que nous utilisons un `TreeMap` dans la classe `TreeIndex`.

Exercice 4

Dans la classe `AndQueryEngine`, écrivez une méthode `private List<Posting> processUnion (List<Posting> list1, List<Posting> list2)` qui calcule l'union (ensembliste) des deux listes passées en paramètres. Ces deux listes sont triées, et la liste résultat doit l'être elle-aussi. Vous devez obligatoirement implémenter l'algorithme vu en cours, qui est optimal.

> Aidez-vous de la correction de l'Exercice 8 de l'examen de TP 2016/17, qui reprend lui même l'Exercice 10 du TP 3.

Exercice 5

Supprimez les classes `AndOrQueryEngine`, `RankingQueryEngine`, et `RankingEvaluator`, dont on n'aura pas besoin au cours de cet examen. Effacez également la classe de test (qui devrait normalement s'appeler `Test1`).

Dans la classe `Configuration`, supprimez le champ `computingScores` ainsi que ses getter et setter. Adaptez les méthodes `FileTools.getIndexFile` et `getPerformanceFile`, en supprimant la partie du code utilisant `Configuration.isComputingScores`.

1. Cette méthode sera modifiée plus en profondeur lors de l'examen.