Projet de Recherche Documentaire

Chaïmaâ Touhami
Timothée Afonso
12/10/2022

Master 1
Informatique

1 INTRODUCTION

Nous avions pour projet de réaliser un programme permettant de faire des recherches de documents par requêtes d'un corpus donné. Pour réaliser un tel système, nous avons dû réfléchir à comment les documents et les requêtes seront traités pour les faire correspondre. Pour cela, plusieurs procédures ont été développé.

Nous précisons que le programme a été écrit en C++

2 PARSING

Le corpus à notre disposition fut constitué de documents formatés sous forme XML. Nous avons pu ainsi, grâce à la librairie RapidXML, récupérer les différentes informations de chaque document pour les traiter et les stocker.

2.1 SUPPRESSION DE CARACTERES SPECIAUX ET TOKENIZATION

Afin d'effectuer un traitement sur des mots cohérents, il faut supprimer les caractères spéciaux, comme la ponctuation par exemple. Tous les caractères non alphabétiques sont remplacés par un espace. L'espace, lui, sera supprimé lors de la phase de tokenisation. Durant cette étape, on en profite pour transformer toutes les majuscules en minuscule.

Une fois les caractères spéciaux supprimés, nous obtenons une chaine de caractères contenant uniquement des caractères alphabétiques et des espaces. La phase de tokenization permet de transformer cette chaîne de caractères en une liste de mots où chacun d'entre eux correspond à un token. Pour l'obtenir, on découpe la chaîne grâce aux espaces.

2.2 UTILISATION D'UNE STOP LIST

Certains mots n'apportent aucune information sur les documents, car ils sont massivement présents dans le corpus. Pour les supprimer, on possède un fichier texte contenant tous les mots inutiles. On parcourt ce fichier ainsi que notre liste de token. Le token est supprimer de la liste dès qu'il est aperçu dans le fichier.

2.3 STEMMING

Si plusieurs mots sont écrits différemment dans les documents, il est nécessaire qu'il compte pour le même mot. Pour cela, la phase de stemming va réduire chaque mot à son radical (suppression des temps, du pluriel, etc…). Pour réaliser cette étape, nous avons implémenté le stemmer de Porter.

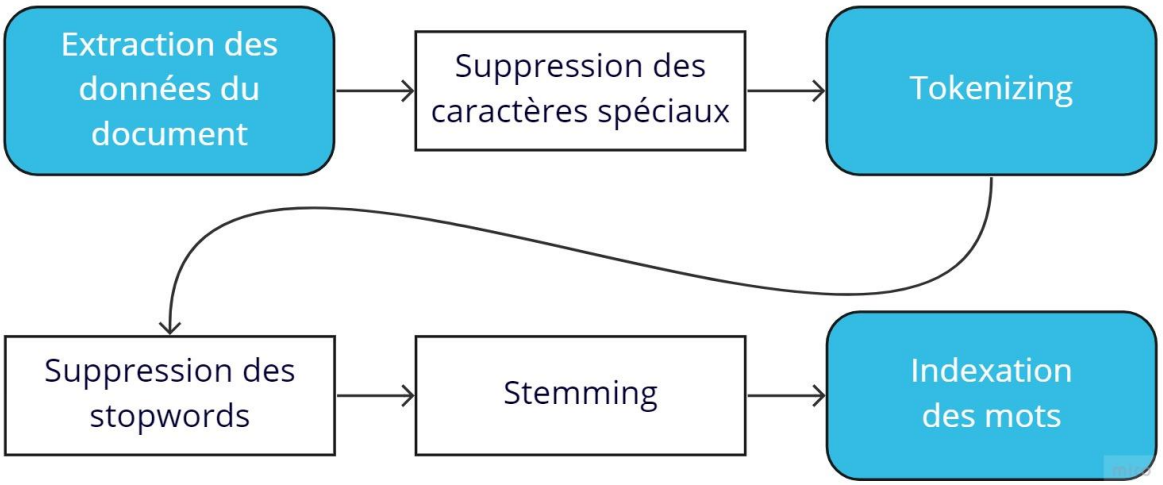


Figure 1 – Diagramme du parsing d'un document

3 INDEXATION

Nous avons fait le choix d'utiliser des B-Tree pour l'indexation, car contrairement aux tables de hachages, la recherche par préfixe est possible. Même si l'utilisation d'un arbre est plus complexe O(log n), le fait qu'on utilise des B-Tree, réduit un peu le temps de parcours. De plus, ce dernier est négligeable à l'échelle de notre système.

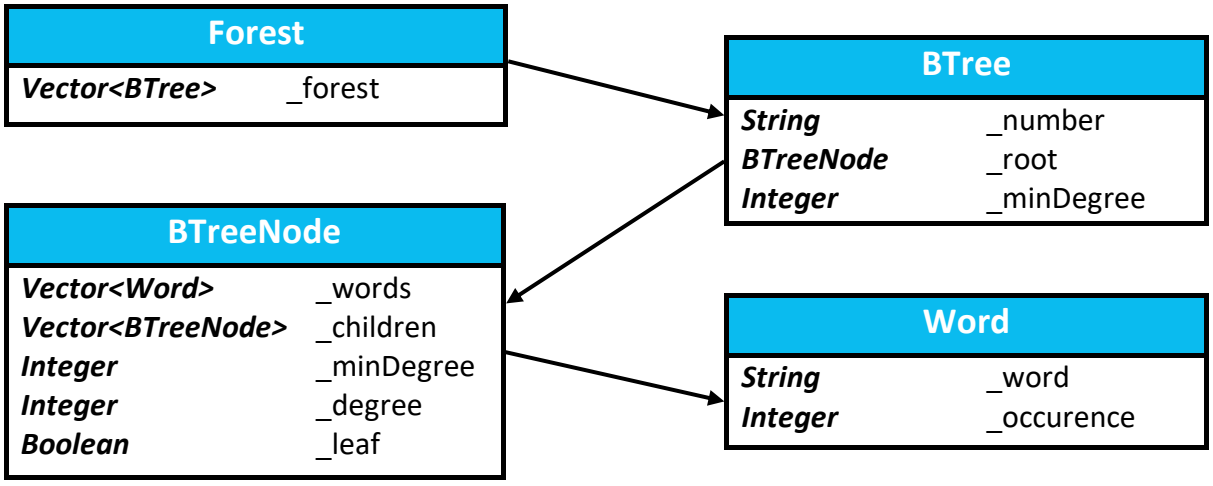


Figure 2 - Diagramme UML Simple des classes pour l'indexage

Le processus d'indexation d'un document commence par la création d'un BTree en passant en paramètre le numéro d'identification du document sur lequel on travaille pour qu'il puisse être retrouvé par la suite, ainsi qu'un degré minimal (arbitraire pour la taille de notre projet).

Ensuite, on ajoute dans cet arbre un à un tous les mots du document sous forme de Word avec _occurence initialisé à 1, de façon à ce qu'ils soient dans l'ordre alphabétique à tout moment. On vérifiera à chaque fois si le _degree de chaque BTreeNode est atteint. Le cas échéant, on séparera en deux le nœud en question pour refaire de la place pour le prochain mot à insérer. De plus, on vérifiera aussi avant d'ajouter un mot si celui-ci n'est pas déjà présent dans l'arbre. Si oui, on ne fera qu'incrémenter l'occurrence du mot déjà existant.

Enfin, lorsque tous les mots du document ont été inséré dans l'arbre, ce dernier est ajouté à la Forest qui stock tous les arbres de chaque document du corpus donné.

4 TRAITEMENT DES REQUETES

Il y a plusieurs façons de traiter la requête :

- Récupérer les documents les plus pertinents en calculant le tf-idf
- Récupérer les documents les plus pertinent en réalisant une requête par préfixe
- Récupérer les documents contenant la requête à l'identique

4.1 REQUETE PAR DISJONCTION (TF-IDF) :

Le traitement avec le calcul du *tf-idf* fonctionne de la façon suivante. Tout d'abord, on réutilise les fonctions de suppression de caractère, tokenization, stopwords et stemming afin d'avoir une liste de token de la requête cohérente avec la liste de token des documents. Pour chaque document, on calcul ensuite le poids *w* de chaque token de la requête en fonction de *tf*, qui correspond au nombre d'occurrence du token dans le document.

$$W(tf) = 1 + log(tf)$$

Ensuite, pour chaque token on calcul *df*, qui est le nombre de document dans lequel apparait le token. Puis on calcul l'*idf* de chaque token qui correspond à la fréquence du token dans tous les documents :

$$idf = log\left(\frac{N}{df}\right)$$ N le nombre total de documents

On peut enfin calculer le *tf-idf* de chaque token :

$$tfidf = w(tf) * idf$$

Pour finir on attribue à chaque document un score. Celui si correspond à la somme des *tf-idf* divisé par le nombre de token de la requête.

On affiche à l'utilisateur les dix documents ayant le score le plus élevé.

4.2 REQUETE PAR PREFIXE :

La requête par préfixe consiste à rechercher les documents avec des mots commencent par le préfixe donné. Une fois ces mots trouvés, le *tf-idf* est aussi calculé. La requête par préfixe se fait en faisant suivre directement un terme par une astérisque. (Ex : democr* pourra correspondre aux mots « democracy » et « democrat ».

4.3 RECHERCHE EXACTE :

La recherche exacte ne nécessite pas de traité la requête comme précédemment. On regarde simplement si la chaine de caractère de la requête est présente tel quel dans un document. La requête avec des termes exact se distingue en étant mise entre guillemets. (Ex : « Ceci est une recherche exacte »)

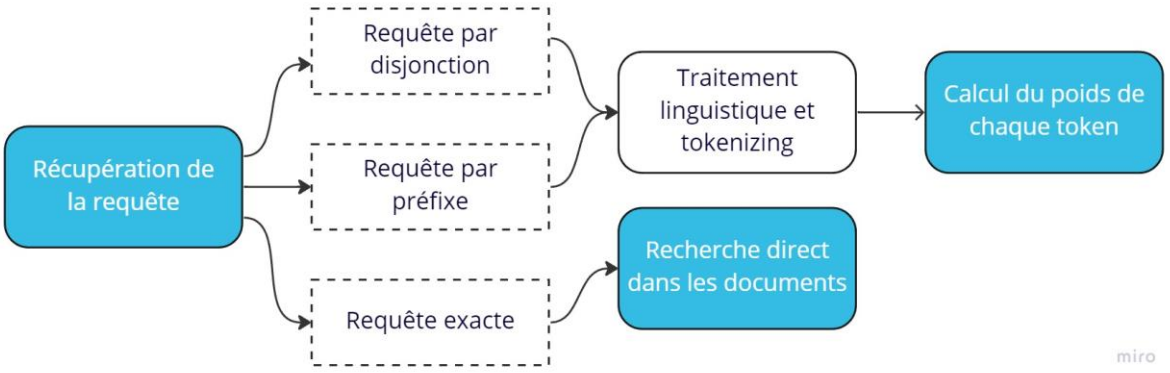


Figure 3 - Diagramme du traitement d'une requête

5 CORRECTEUR

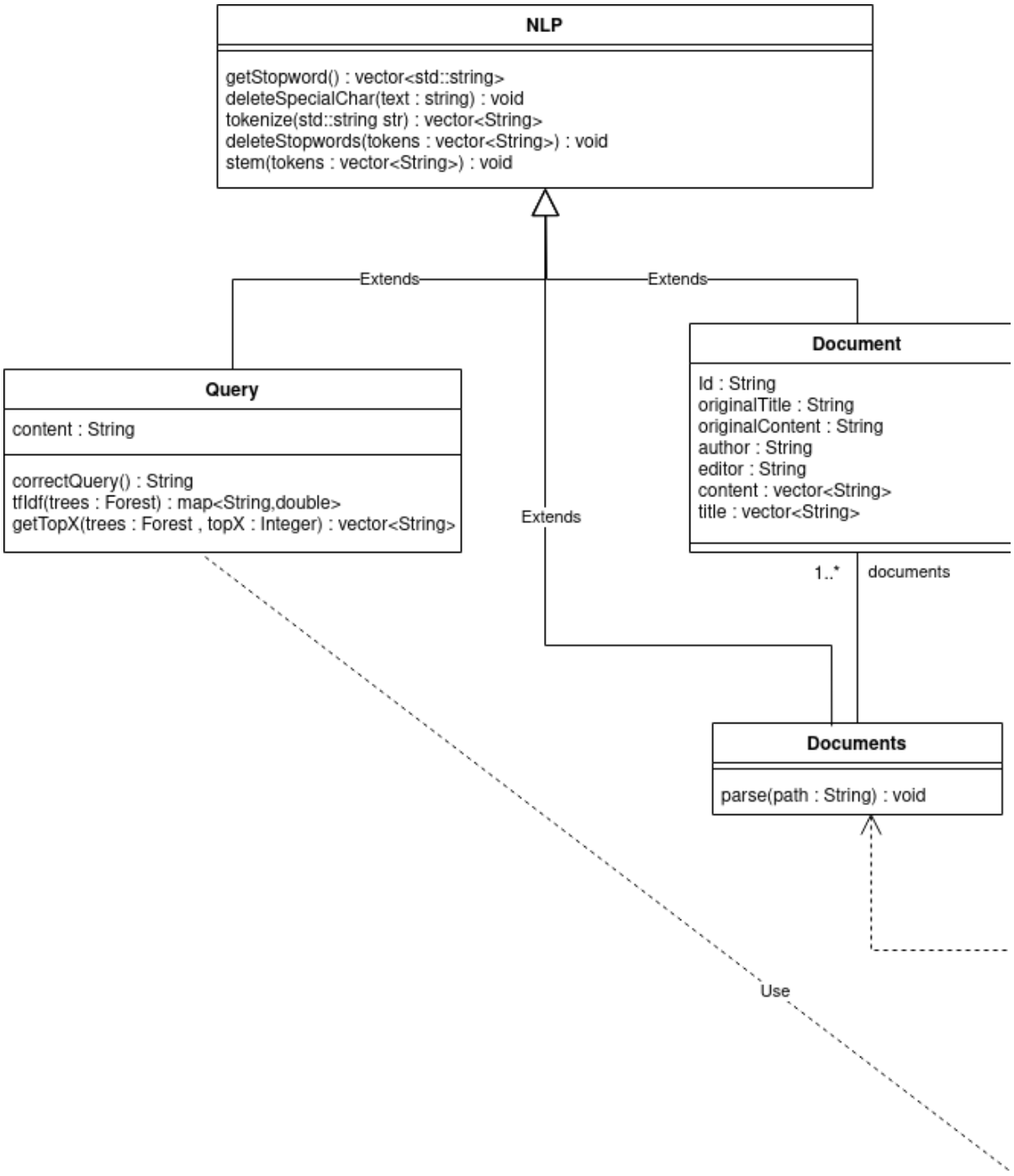
Nous avons également implémenté le correcteur de Peter Norvig. Celui-ci trouve parmi les mots les plus proche de celui mal orthographié, le mot le plus probable d'être le bon. Pour cela le théorème de Bayes est utilisé. On utilise aussi un fichier texte qui est une concaténation d'extraits de livres du domaine public du projet Gutenberg et de listes de mots les plus fréquents du Wiktionnaire et du British National Corpus. On détermine ainsi la fréquence d'apparition de chaque mot parmi les millions de mots présent dans le document. On peut ainsi calculer la probabilité qu'un mot corresponde au mot mal orthographié.

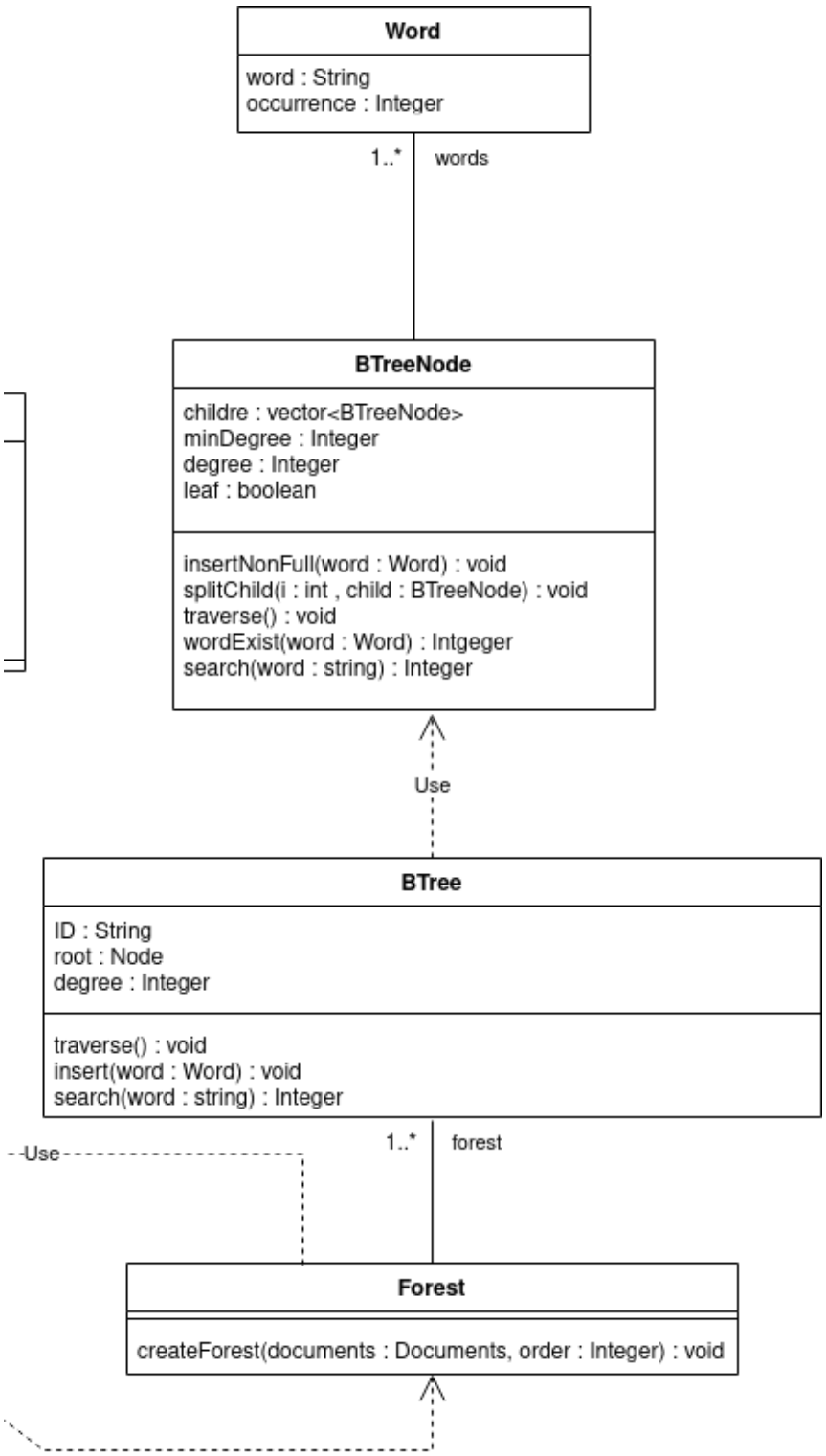
6 POUR ALLER PLUS LOIN

Pour faciliter l'utilisation du système, nous avons créer une interface graphique permettant de taper une requête. Une correction peut être proposée si un mot a été mal orthographié. Les documents trouvés sont affichés sous forme de liste avec leur titre. Chaque élément de la liste est cliquable permettent d'ouvrir une fenêtre et d'afficher toutes les informations du document.

Du fait que nous ayons travaillé sur un corpus de documents, il aurait pu être pertinent d'aussi créer des requêtes sur l'auteur ou l'éditeur de ceux-ci pour des résultats plus précis.

7 DIAGRAMME UML DU PROGRAMME




Word
word : String
occurrence : Integer
1..*
words
BTreeNode
childre : vector<BTreeNode>
minDegree : Integer
degree : Integer
leaf : boolean
insertNonFull(word : Word) : void
splitChild(i : int , child : BTreeNode) : void
traverse() : void
wordExist(word : Word) : Intgeger
search(word : string) : Integer
Use
BTree
ID : String
root : Node
degree : Integer
traverse() : void
insert(word : Word) : void
search(word : string) : Integer
1..*
forest
Use
Forest
createForest(documents : Documents, order : Integer) : void