WEM : Web Mining
Laboratoire n°1
Crawling, indexation et recherche de pages Web

17.03.2017

Objectifs

Ce laboratoire a pour but d'implémenter un moteur d'indexation pour une collection de pages web de votre choix, puis d'utiliser l'index créé pour implémenter un outil de recherche simple.

Les points étudiés dans ce laboratoire seront :

- Crawling de pages web (en utilisant la librairie *crawler4j*[1] fournie)
- Construction d'un index et d'un index inversé
- Pondération des termes
- Implémentation du modèle vectoriel
- Implémentation d'une fonction de recherche

Durée

- 6 périodes. A rendre le **07.04.2017** à 14h00 au plus tard.

Références

- Cours «Web Mining» de Nastaran Fatemi et Laura Raileanu
- Livre «Data Mining the Web: Uncovering Patterns in Web Content, Structure, and Usage» de Zdravko Markov et Daniel T. Larose

Donnée

Le laboratoire se déroulera en deux parties distinctes. Dans un premier temps, vous crawlerez un site web de votre choix à l'aide de la librairie *crawler4j*, vous construirez le/les indexes que vous sauverez sur le disque. Par la suite, vous chargerez en mémoire cet/ces indexes qui permettront à un utilisateur d'effectuer des recherches textuelles.

Vous trouverez sur la Figure 1, un schéma représentant l'architecture de l'application. Nous reviendrons ensuite en détail sur vos tâches à réaliser pour chaque étape.

[1] https://github.com/yasserg/crawler4j

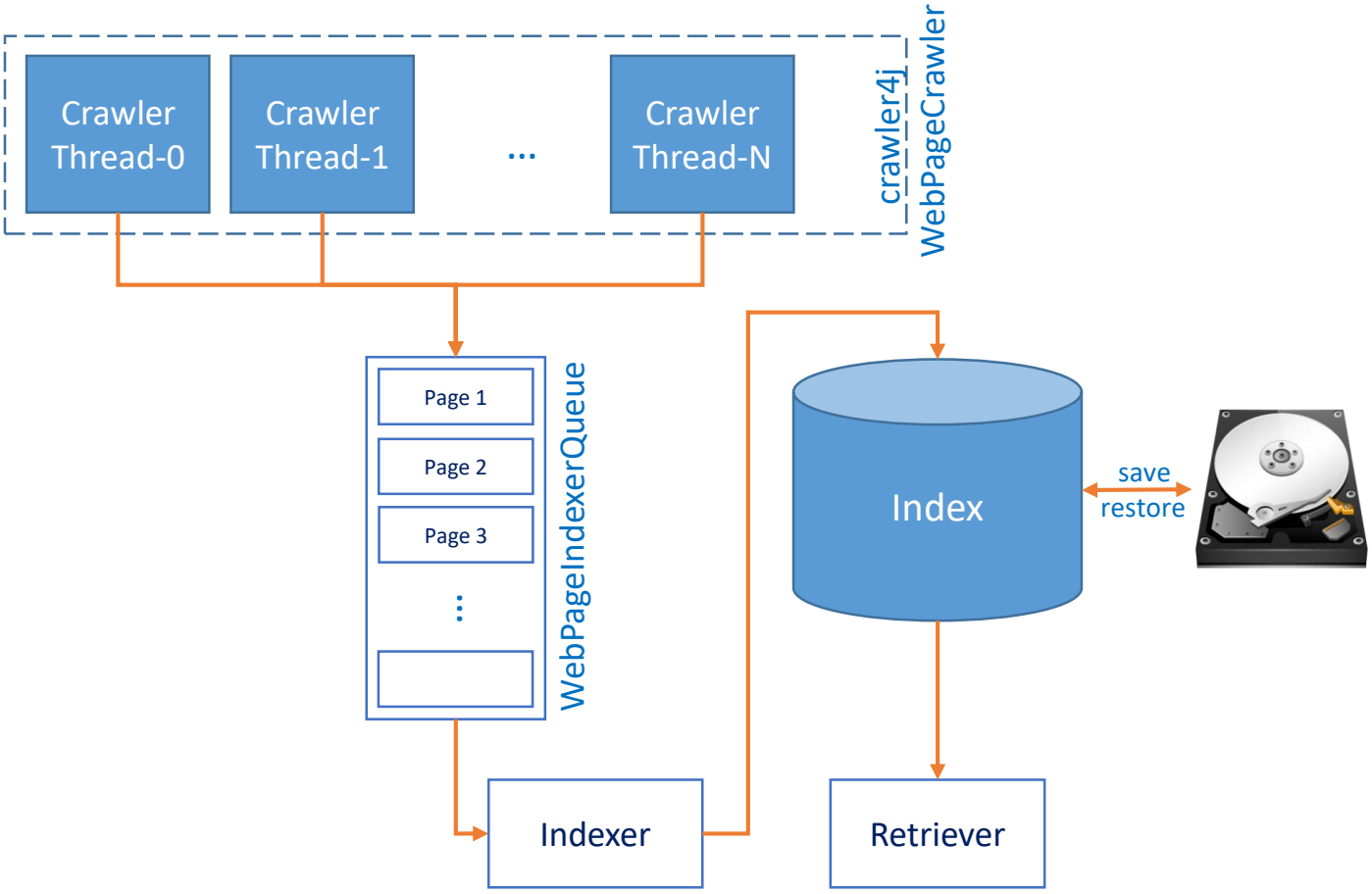

Figure 1 – Schéma de l'application

1. Crawler

WebPageCrawler extends edu.uci.ics.crawler4j.crawler.WebCrawler

Nous vous fournissons cette classe, vous devez juste implémenter une méthode.
Les threads initiés par *crawler4j* instancieront et utiliseront cette classe pour poster les résultats de leur visite. Cette classe possède une référence statique vers une *WebPageIndexerQueue* qui servira de file d'attente à l'indexer. Cette classe possède 2 méthodes principales :

public boolean shouldVisit(Page referringPage, WebURL url) **A implémenter**

Pour chaque lien rencontré par le crawler durant sa visite, il demandera à cette méthode si la page doit être récupérée. A vous de faire en sorte que les ressources vraisemblablement non-supportées (non-textuelle) ou inutiles ne soient pas visitées. **Vous limiterez aussi la visite uniquement au domaine ciblé.**

public void visit(Page page) **Peut être modifiée/améliorée**

Cette méthode va mettre dans la queue les données des pages visitées. Vous pouvez ici décider de limiter l'indexation uniquement aux formats supportés, tous les cas ne pouvant être évités avec la méthode précédente.

Quelques remarques/précision

- Le crawler reste en attente un certain temps en fin de crawling (jusqu'à 30 secondes). Ceci est un comportement normal.
- Lors de ce laboratoire vous ne devez pas vous soucier des liens entre les pages. Ils seront utilisés dans le prochain laboratoire.
- Le site web que vous choisirez ne devra pas contenir plus de 500 pages. Un site trop grand risque de faire exploser la taille de l'index en mémoire. En effet, les recherches seront effectuées dans un index chargé en mémoire. Vous pouvez limiter le nombre de pages visitées avec le paramètre *maxPagesToFetch*.

2. Indexation

Vous devez ici implémenter votre indexeur qui aura pour interface (non-modifiable) *ch.heigvd.wem.interfaces.Indexer* que vous instancierez dans la méthode *crawl()* de *Labo1*.

Votre indexeur utilisera une sous-classe de *ch.heigvd.wem.interfaces.Index*, que vous définirez librement et qui représentera l'index en mémoire. Les seules contraintes sont qu'elle doit hériter de la classe abstraite *Index* et que tous ses attributs soient sérialisables afin de pouvoir l'écrire dans un fichier sur le disque. Elle devra certainement aussi permettre de pouvoir récupérer une table de correspondance id/pages, afin de vous permettre de présenter les résultats.

L'indexation d'un document peut être décomposée comme suit :

- Il faut commencer par extraire les tokens du contenu reçu pour l'indexation. Il faut donc séparer le String reçu en une séquence de mots. Les espaces et la ponctuation constituent les séparateurs. On réservera un traitement particulier à l'apostrophe : une apostrophe en début ou en fin de mot devra être ignorée, mais maintenue si elle se situe au milieu d'un mot. Par exemple :
  - *I **won't** talk* → [i, **won't**, talk]
  - *My **parents'** house* → [my, **parents**, house]
- Il faut ensuite procéder à l'élimination des « stop words », à savoir le filtrage des mots qui n'ont pas de signification. Ces mots sont fournis dans le fichier *common_words*. Pour ce filtrage, il ne faut pas tenir compte de la casse ;
- La lemmatisation ou la troncature des mots significatifs pourrait être réalisée ici mais elle ne vous est pas demandée ;
- On stocke ensuite, pour chaque document traité, les mots qu'il contient (en minuscules) ainsi que leur fréquence ;
- On génère également un index inversé contenant, pour chaque mot, les *id* des documents les contenant ainsi que leur fréquence. L'identifiant d'un document est généré automatiquement et est présent dans les métadonnées.

Une fois tous les documents indexés, vous devrez implémenter deux types de pondération pour le contenu de la collection de pages Web récupérées. Ces pondérations impliquent un nouvel index et un nouvel index inversé. Vous stockerez les 2 pondérations dans votre index.

- **Pondération par fréquence normalisée :**
  $w(t_i) = freq_i / \max_j(freq_j)$

- **Pondération par tf*idf normalisée :**

$$tfidf(t_i) = \log_2(freq_i + 1) \cdot \log_2(N / n_i)$$

$$w(t_i) = tfidf(t_i) / \max_j(tfidf(t_j))$$

Où $freq_i$ est la fréquence d'occurrence du terme t_i dans un document, N est le nombre de pages web dans la collection, n_i le nombre des pages web qui contiennent le terme t_i et $max_j()$ est la valeur maximale de la pondération dans une page web.

Votre indexeur devra, au minimum, implémenter les méthodes suivantes :

public void index(Metadata metadata, String content)

Méthode appelée par *WebPageIndexerQueue* pour chaque page visitée avec les métadonnées et le contenu brut de la page. Vous réaliserez ici le traitement sur les données textuelles brutes, tokenisation, élminitation des stop-words, etc.
Faut-il aussi indexer certaines métadonnées ?

public void finalizeIndexation()

Méthode appelée lorsque toutes les pages auront été visitées. Cela vous permet de finaliser le calcul des pondérations.

public Index getIndex()

Méthode pouvant être appelée, uniquement après *finalizeIndexation()* pour pouvoir sauver votre index sur le disque.

3. Retriever

Le retriever permet d'effectuer des recherches textuelles et d'avoir en retour une *Map* triée avec les résultats les plus pertinents en premier. Votre solution devra implémenter la classe abstraite *ch.heigvd.wem.interfaces.Retriever*, il est suffisant de définir uniquement la méthode *executeQuery()*, par contre il se peut que vous ayez besoin des autres méthodes pour ce faire.

Vous devez implémenter le modèle vectoriel avec le calcul de similarité par cosinus. La requête sera entrée sous une forme libre et sera traitée comme pour les pages Web (suppression des mots vides, ponctuation, etc.).

Votre programme doit donc accepter une requête et fournir une liste ordonnée de pages Web comme réponse (en faisant apparaître les valeurs de similarité cosinus). Il devra en outre laisser choisir à l'utilisateur le type de pondération utilisé pour la recherche (et donc quel index utiliser).

4. Programme principal

Finalement, il s'agit de compléter la classe *Labo1*. Cette classe contient le point d'entrée de l'application. Elle est chargée d'orchestrer les différentes opérations à réaliser. Elle doit donc effectuer l'instanciation des différents objets, lancer l'indexation de la collection de pages Web ou le chargement des indexes depuis le disque et enfin la démonstration du bon fonctionnement des méthodes de recherche. Nous vous fournissons déjà une ébauche que vous devrez compléter, vous n'êtes pas obligés d'implémenter un dialogue avec l'utilisateur, mais cela vous facilitera la tâche pour vos tests.

Rendu/Evaluation

Vous remettrez sur *Moodle* un projet Eclipse zippé contenant les sources, libraires utilisées, fichiers d'entrée, etc. Attention aux indexes qui peuvent être volumineux et qu'il n'est pas nécessaire de rendre.

Nous ne demandons pas de rapport, mais veuillez bien commenter votre code, au minimum les étapes importantes, les choix que vous avez dû faire ainsi que tout ce dont vous jugerez utile.

Vous pouvez discuter entre les groupes mais il est strictement interdit d'échanger du code.

Adresse E-Mail de l'assistant : fabien.dutoit@heig-vd.ch

Bonne chance !