Université Paris Diderot

Méthodes et Algorithmes pour l'accés à l'Information Numérique (MAIN)

TP2 : Programmer un moteur de recherche

Réalisé par :
Kheireddine Ouelaa

25 mai 2015

Résumé

Le but de ce TP est de programmer un moteur de recherche, l'utilisateur entre une requête (un mot), le moteur lui renvoie la liste ordonnés des pages contenant ce mot (les pages de rang les plus élevé en premier).

Le moteur est programmé en implémentant un algorithme que nous détaillerons dans ce rapport en utilisant les résultats de moteurs de recherches déjà connus. Dans notre programme nous utilisons 3 de ces moteurs : Google, Bing, Yahoo.

Nous utilisons Java pour l'implémentation du programme, avec l'aide d'une bibliothèque JSoup qui fournit une API très pratique pour l'extraction et la manipulation des donnés HTML.

Pour l'installation et l'execution du programme, penser à lire le README associé à ce rapport.

Table des matières

1 Moteur de recherche

1.1 Principe et algorithme

L'idée ici est d'associer à chaque sommet d'un graphe du web une liste de 5 mots, pris parmi un dictionnaire de 100 mots.
Les sommets en question sont les sommets d'un graphe du web, ou on a calculé son PageRank (Algorithme détaillé dans le TP1).

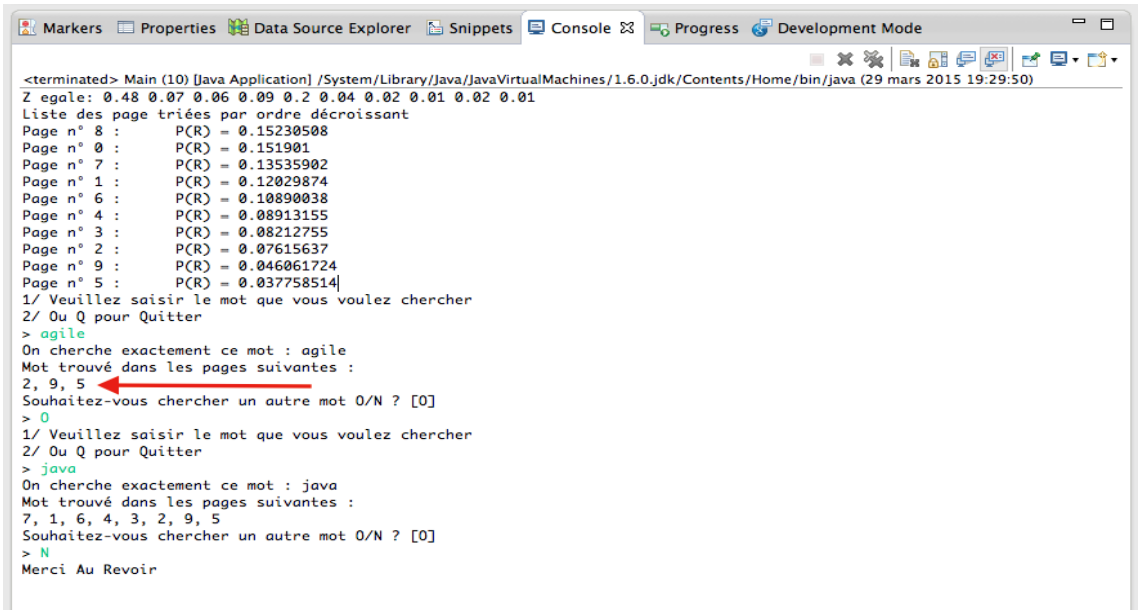
Pour ce faire, nous avons crée une classe Database qui nous permet de lire un fichier dictionary.txt qui contient une liste de 100 mots générés aléatoirement, on range tous les mots dans une liste qui nous permettra par la suite de tirer cinq mots au hasard et de les attribuer au noeud.

Le programme récupère ensuite le vecteur généré par le calcul du PageRank, qu'on triera par ordre décroissant.

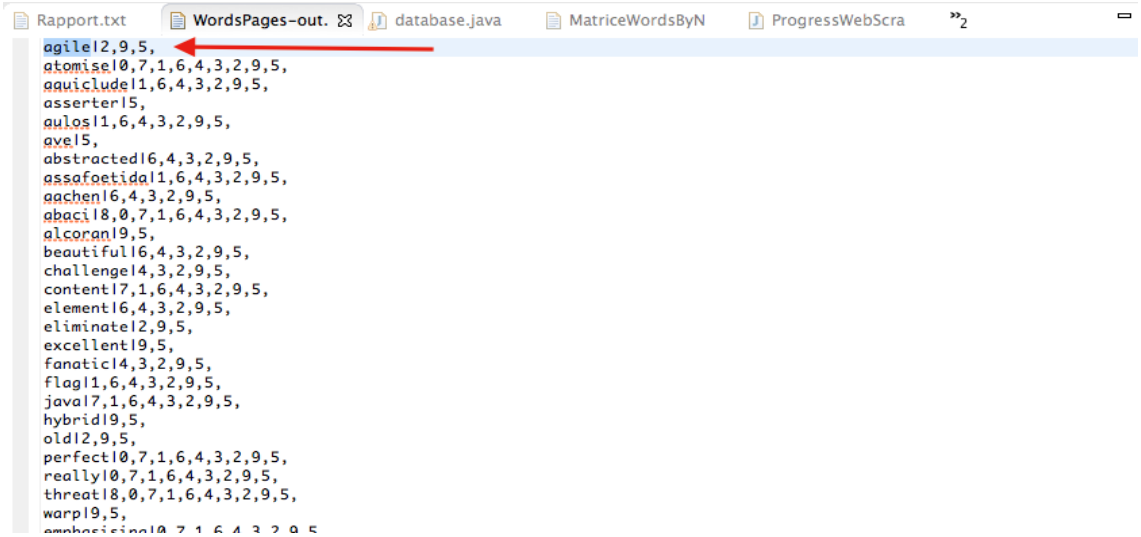
L'utilisateur demande un mot parmi les mots du dictionnaire, le programme lui renvoie les pages contenant ce mot **par ordre croissant selon la valeur du PageRank**

1.1.1 Exemple d'execution

Dans la simulation qui suit, l'utilisateur cherche le mot : "agile".



On vérifie dans l'image qui suit que la requête demandé se trouve en effet dans les pages indiqués.



1.2 Complexité et performance

Le programme s'execute en temps : $T \in O(n)$
En effet la complexité du PageRank est de $O(n+m)$ (n=taille de matrice et m=valeurs non-nuls) $\Rightarrow O(n)$
+ la complexité de l'association des 5 mots pour chaque sommet $\Rightarrow O(5 \times n)$
+ la complexité du parcours de la HashMap qui est de l'ordre de $O(1)$
$\Rightarrow T \in O(n)$

1.3 Amélioration : requête de plusieurs mots (Borda ?)

L'idée est de favoriser les pages qui sont en commun pour les deux mots (union) en gardant l'ordre de leur apparition. On peut utiliser la méthode Borda décrite plus bas dans le rapport (juste une intuition).

Exemple :
On fait une recherche sur ces deux mots : "agile alcoran", on obtient ce résultat :
agile|2,9,5,
alcoran|9,5,
Le moteur devra afficher les pages 9,5,2 dans cet ordre.

2 Agrégation des résultats de moteurs de recherche

2.1 Posons une question aux différents moteurs : "Comment voter blanc ?"

En posant cette au différents moteur de recherche et en prenant par exemple le premier lien sur Yahoo celui de Wikipédia (fr.wikipedia.org/wiki/Vote_blanc), on constate que celui ci ne vient qu'à la 7ème place sur Google et à la 2ème place sur Bing.

2.2 Description technique : Méthode Borda

Pour agréger les résultats donnés par les moteurs de recherches, nous utiliseront la méthode Borda, qui est une méthode très ancienne[1], beaucoup utilisé dans le système électoral et mode de scrutin.
La méthode consiste à calculer un score pour chaque lien trouvé dans les différents moteurs, à partir de ces scores on peut avoir notre propre classement dans le nouveau moteur de recherche en triant les scores calculés avant.

2.2.1 Algorithme

L'idée de l'algorithme est donc d'appliquer la méthode Borda décrite plus haut. Pour ce faire on parcourt des ArrayList<Liens> (déjà remplis avec les données de recherches de chaque moteur). On calcule à chaque fois le score de chaque lien pour le mettre dans une HashMap<Lien,Score> (nous expliquerons ce choix d'implémentation just après), la clé de celle-ci sera le lien en question et sa valeur sera le score calculé.
Pour le calcul du score nous nous exposons à deux cas :

1. Le cas ou le lien n'a pas été déjà trouvé, nous calculons le score avec cette equation :
   :
   Score(L) = (M-1) × (R+1) +C
   Où : **L**=Lien, **M**=nombre de Moteurs, **R**=nombre de Résultats traités, **C**= Classement actuel du lien.

   Explication : Au début on ne connait pas encore si le lien est présent dans les recherches d'autres moteurs ou pas, on suppose donc que ce dernier n'y est pas dans les autres moteurs d'où le $(M-1) \times (R+1)$ et on l'additionne à son classement actuel.

   Exemple : Si on a le mot "B" à la 2ème place, et que l'on a 3 moteurs de recherches et 4 résultats, on aura : $(3-1) \times (4+1) + 2 \Rightarrow Score(" B") = 12$
2. Le cas ou le lien a été déjà trouvé dans d'autres moteurs, nous calculons le score avec cette equation :

1. Utilisé déjà l'an 105. Elle a été formalisée en 1770 d'après Wikipédia, 1784 d'après d'autres sources

Score(L) = ScoreMap(L) - (R+1) + C
Où **ScoreMap(L)** = le score déjà calculé et stocké dans la Map

Explication : Si le score a été déjà calculé, donc la supposition que ce lien n'y figure pas dans la liste de la recherche pour ce moteur est fausse d'où le $-(R+1)$

Exemple : Pour reprendre l'exemple précédent, si le mot "B" figure à la premiere place dans la liste de ce moteur on aura : 12(score de "B" avant)−5 (nombre de résultat + 1) +1 (classement)
$\Rightarrow Score(\text{"}B\text{"}) = 8$ (on remarque que le score diminue à chaque fois que le mot/lien est présent dans le résultat de la recherche)

2.3 Programme Java et exemple d'execution

Comme mentionné dans le résumé du rapport, le programme a été élaboré en Java avec une bibliothèque JSoup qui fournit une API pour l'extraction et la manipulation des données.
Le programme demande à l'utilisateur (en ligne de commande) une requête (d'un simple mot à une phrase), le programme récupère les résultats de recherche depuis les différents moteurs de recherches, applique l'algorithme décrit plus haut, et affiche à l'utilisateur le nouveau classement des liens.

Voici un petit exemple d'execution :
Google : [A, B, C, F]
Bing : [B, A, C, K]
Yahoo : [B, E, C, K]
Meta-moteur : {B=4, A=8, C=9, E=12, K=13, F=14}

2.4 Difficultés

La difficulté résidait dans l'implémentation de la méthode Borda en programme Java.
En effet au début nous avions pensé à l'algorithme naïf, qui calcule le score des liens en sommant le classement de chaque lien dans chaque moteur de cette façon : $\sum_{m=1}^{M} C_m$ où M est le nombre de moteurs, et C le classement du lien.

Cette méthode s'est avérée être très gourmande en terme de complexité, puisque pour chaque entrée (lien), on est obligé de parcourir R fois au moins pour les (M-1) listes. (R est le nombre de résultats et M le nombre de moteur)
$\Rightarrow T \in O(n^{M-1})$

2.5 Complexité et performance

La complexité de notre algorithme est de l'ordre de
$\Rightarrow T \in O((M-1) \times R)$ ....Beaucoup mieux que l'algorithme naïf.

Cette amélioration se traduit par l'utilisation des HashMap pour stocker les scores de chaque lien, la clé de la map est donc le lien et la valeur est son score. L'utilisation de cet objet améliore l'algorithme puisque le parcours de la map se fait en $O(1)$.