Projet

Les dates importantes :
- 17 juin, présentation du projet,
- du 18 juin au 24 juin, projet à plein temps
- 1er juillet soutenances

1 Contexte : Recherche de Documents

Objectif : Le projet a pour objectif de réaliser un outil de recherche dans une base de documents. Une requête est composée de mots et d'opérateurs booléens, le résultat est l'ensemble des documents de la base qui "répondent" à la requête. Le moteur de recherche doit être accessible au travers du réseau.

La base de documents : La base de documents est constituée de fichiers au format texte correspondant aux documents eux-mêmes, ainsi qu'un fichier servant d'indexation de la collection.

Nous vous fournissons (voir plus loin) un programme permettant de charger directement le fichier d'index (sous forme de tables de hachage/dictionnaires) dans votre programme.

Vous disposez également d'une collection "jouet" contenant 3 documents :

- *Doc1* : contient le texte "langage python langage java".
- *Doc2* : contient le texte "programmation java".
- *Doc3* : contient le texte "programmation python".

Les requêtes : Différents types de requêtes peuvent être envisagés. Sous sa forme la plus simple une requête est composée d'un mot : on cherche tous les noms des documents dans lesquels ce mot apparaît. La requête peut aussi être composée d'une liste de mots. On peut considérer que cette liste de mots constitue un document de référence et l'on cherche les documents de la base qui s'en rapprochent le plus suivant la mesure du cosinus. Enfin, on peut ajouter des opérateurs booléens entre les mots. Par exemple, si l'on recherche les documents dans lesquels apparaît le mot $fleurs$ et soit le mot *haine* soit le mot *amour*, la requête correspondante est : $fleurs\ et$ (*haine ou amour*).

Indexation de la base : Pour ne pas avoir à parcourir tous les textes à chaque requête, la base de textes/documents est indexée. Cette opération est précédée d'une normalisation des graphies et les mots "outils" ne sont pas indexés. L'index permet de savoir, pour chaque mot, le nombre d'occurrences de ce mot dans chacun des documents.

Ce travail d'indexation a été réalisé à l'aide du fichier `indexerNoStem.py` qui vous est fourni.

La collection qui vous est fournie contient le fichier d'index suivant (le caractère séparateur est la tabulation '\t') :

- à chaque début de ligne, un mot présent dans la collection
- puis, pour chaque document contenant ce mot, le nom du document suivi du nombre du mot dans le

document ; le caractère " :" sert de séparateur entre nom de document et le nombre d'occurrences. Exemple :
`programmation Doc2:1 Doc3:1`
(i.e., le mot `programmation` apparaît dans les documents `Doc2` et `Doc3` avec à chaque fois 1 pour nombre d'occurrences).

Afin de faciliter votre travail, nous vous proposons de retenir dans un premier temps une indexation sous forme de tables de hachage :

- une table unique dont les clés sont les mots du corpus et dont les valeurs sont elles-mêmes des tables de hachage ;
- pour chaque mot, la table de hachage associée a pour clés les noms des documents qui contiennent ce mot, et pour valeurs les nombres d'occurrences correspondants.

Vous utiliserez pour cela la classe `Hashtable` de Java et les primitives associées, ou bien les dictionnaires de python.

Le programme fourni et structures de données : Il vous est fourni un programme `IX` qui lit le fichier d'indexation et le charge en mémoire sous la forme décrite ci-dessus. Deux implémentations sont disponibles : `IX.java` (et `Test.java` pour un programme de test) en java et `IX.py` pour le code correspondant en python.

Les structures de données fournies sont :

- un "dictionnaire de documents". `dicoDocs` est un `Set<String>` (en java) ou un `set` (en python) contenant les noms de tous les documents de la base ;
- un "index" d'occurrences. `wordFreqInDoc.get(w).get(d)` est le nombre d'occurrences du mot `w` dans le document `d`.

Ce qu'il faut faire :

Parmi toutes les questions posées ci-dessous, toutes ne sont pas obligatoires, et elles sont relativement indépendantes. Vous pouvez décider d'établir des priorités, d'en zapper certaines pour pouvoir consacrer plus de temps à d'autres, etc.

Elles suivent cependant une certaine progression logique :

- d'une part parce que les premières questions facilitent les suivantes ;
- d'autre part parce que certaines des dernières questions n'ont réellement de sens qu'après avoir traité les précédentes.

Si vous coincez sur une question "avancée", il vous est donc conseillé de revoir plus attentivement les parties initiales que vous auriez pu survoler.

2 Prise en main (2h maximum)

Q1. Compiler, exécuter, essayer, lire, comprendre les programmes fournis....

3 Algorithmique et recherche d'informations

Q2. L'utilisateur donne un mot (la requête) ; vous devez afficher l'ensemble des documents qui contiennent ce mot.

Q3. L'utilisateur donne un mot (la requête) ; vous devez afficher la liste de tous les documents qui contiennent le mot en classant cette liste par ordre décroissant du nombre d'occurrences.

Q4. L'utilisateur donne un ensemble de mots (la requête). Considérant que cette requête est un document, utiliser la mesure de similarité du cosinus pour retourner à l'utilisateur les (k) documents qui correspondent le mieux à sa requête. Le résultat devra être fourni de manière ordonnée (les premiers documents étant ceux avec le cosinus le plus élevé).

Rappel sur le cosinus : La formule vue en cours :

$$cos(X,Y) = \frac{\sum_{i=1}^{Nb\_Mots} x_i y_i}{\sqrt{\sum_{i=1}^{Nb\_Mots} x_i^2 \sum_{i=1}^{Nb\_Mots} y_i^2}}$$

Q5. On considère toujours que la requête est un ensemble de mots. Retourner à l'utilisateur l'ensemble des documents qui contiennent au moins l'un de ces mots clefs (c'est l'équivalent d'un "ou" entre les mots, écrire une procédure *op_ou*(...)).

Q6. On considère toujours que la requête est un ensemble de mots. Retourner à l'utilisateur l'ensemble des documents qui contiennent ces mots clefs (c'est l'équivalent d'un "et" entre les mots, écrire une procédure *op_et*(...)).

4 Analyse syntaxique

Une requête est une expression composée de mots (des chaines de caractères), des opérateurs *ou* et *et*, et de parenthèses. Les opérateurs sont associatifs à gauche, l'opérateur *et* est plus prioritaire que l'opérateur *ou*.

Q7. Décrire ce langage à l'aide d'une grammaire hors-contexte.

Q8. Rendre la grammaire LL(1) si elle ne l'est pas.

Q9. Ecrire un analyseur syntaxique pour ce langage. Si vous programmez en java, vous pourrez utiliser `JavaCC` ou `antlr4`. Si vous programmez en python, seul `antlr4` peut être utilisé.

Q10. Sur la base des programmes précédents (paragraphe 3), écrire un programme qui permette de retourner l'ensemble des documents qui satisfont une requête booléenne, dans laquelle les mots sont reliés par les opérateurs booléens *et* et *ou*.

Q10bis. Compléter les programmes précédents ainsi que votre analyseur syntaxique pour traiter les requêtes booléennes comportant l'opérateur *non* (qui permet de sélectionner les documents ne correspondant *pas* à une requête).

5 Client/serveur

Nous vous conseillons de décider dès le début de votre projet de programmation de traiter ou non cette partie.

Pour proposer une version distribuée de votre application de recherche de documents, vous aurez besoin des sujets traités en Systèmes & Réseaux. En particulier, vous travaillerez avec des *processus*, des *threads* et des *sockets*. Les supports de cours et les TPs relatifs à ces sujets sont disponibles sur la page de la matière `https://du-isn.gricad-pages.univ-grenoble-alpes.fr/2-sr`.

Dans le cadre de votre projet, vous êtes invités à traiter la première des deux questions qui suivent.

La deuxième question est une question bonus. Elle implique la manipulation du protocole HTTP que vous n'avez pas eu le temps de traiter en cours. Si vous êtes intéressés ou tout simplement curieux par cette partie, discutez-en avec vos enseignants.

Q11. Intégrer le moteur de recherche au sein d'une application serveur, interrogeable à travers le réseau par une application cliente (via des sockets TCP/IP). La spécification du protocole d'interaction client-serveur est laissée libre. Si le format du protocole est en mode texte, vous pourrez éventuellement utiliser telnet comme application cliente (avant / plutôt que de coder votre propre application cliente).

Q12. Bonus pour les très motivés. Intégrer le moteur de recherche au sein d'un serveur Web (protocole HTTP), sous la forme d'un script CGI. Pour cela, vous pourrez vous inspirer du mini serveur Web et des exemples de scripts CGI fournis en annexe du cours. De préférence, l'exécution du script d'interrogation sera lancée via un formulaire HTML et produira une page de résultats également au format HTML (avec des hyperliens vers les documents trouvés).