Moteur de recherche pour Wikipédia Mathématiques

L'objectif de ce TP est de programmer un petit moteur de recherche pour `https://wiki.jachiet.com`. Ce site héberge un extrait de Wikipedia francophone avec uniquement des articles de thématique "mathématiques". Ce moteur de recherche devra fonctionner en 4 phases indépendantes, le résultat de chaque phase étant stocké uniquement dans une base de données :

1. crawling du site
2. extraction de l'index inversé de l'inverse document frequency
3. calcul du PageRank
4. requêtage de notre moteur de recherche

Si vous souhaitez rendre ce tp, il vous est demandé de rendre une archive (Zip ou autre) sur le Moodle qui doit contienir au moins 3 fichiers Python3 nommés `index.py`, `pagerank.py` et `query.py` correspondant respectivement aux phases 2, 3 et 4. En plus de ces fichiers il vous est aussi demandé de mettre un fichier (txt ou pdf) expliquant ce que vous avez ou non fait et répondant aux questions posées dans cet énoncé. Vos fichiers sources doivent être lisibles, c'est à dire que les variables sont nommées en décrivant ce qu'elles stockent, les fonctions en rapport avec leur effet et chaque grande étape de votre algorithme doit être décrite avec des commentaires.

1 Première phase : crawling du site

Pour vous éviter de devoir crawler le site vous-même (ce que vous n'apprendrez à faire correctement que dans quelques leçons) je vous mets à disposition le résultat du crawl sous la forme d'un fichier SQLite qui contient deux tables : "webpages" et "responses". La table "responses" contient deux colonnes de type TEXT : une colonne "queryURL" avec l'URL que notre crawler a demandé et une colonne "respURL" avec l'URL que notre crawler a reçu. Il est important de bien comprendre la différence le 'queryURL' et le 'respURL'. Il peut y avoir plusieurs sources de différence entre ces deux URLs :

— cela peut être une question d'encodage ("Géométrie" qui devient "G%C3%A9om%C3%A9trie")
— parce qu'une URL a définitivement changé en une autre (code HTTP 302) ("/Base_2" qui devient "/Système_binaire" par exemple)
— parce qu'une URL renvoie temporairement vers une autre (code HTTP 301) ("/random" qui devient "/Kurt_Gödel").

Un moteur de recherche idéal devrait traiter ces cas différemment mais parce que nous faisons un exercice nous allons simplement tout traiter uniformément. Pour cela nous allons indexer les documents par leur URL de réponse et deux documents qui ont la même URL de réponse seront considérés égaux.

La table "webpages" contient 2 colonnes de type TEXT : une colonne 'URL' et une colonne 'content' avec le contenu des pages Web retourné par le serveur. Attention, l'URL stockée dans la table correspond à l'URL donnée par le serveur.

Pour la première phase, il ne vous est pas demandé de faire le crawling, vous devez simplement télécharger le fichier `crawl.zip`, le dézipper puis de regarder son contenu. Si vous avez installé `sqlite3`, vous pouvez l'explorer avec la commande `sqlite data.db` puis en utilisant SQLite, par exemple depuis Python avec le code suivant :

```
import sqlite3
conn = sqlite3.connect("data.db")
conn.execute("SELECT * FROM webpages LIMIT 1")
```

Ne pas oublier de faire `conn.commit()` à la fin de votre fichier, après les commandes qui modifient la base de données. Voir la documentation à l'adresse : `https://docs.python.org/3/library/sqlite3.html`.

Télécharger l'archive `template.zip` qui contient 4 fichiers python (`shared.py`, `index.py`, `pagerank.py` et `query.py`) ainsi que `data.db`, la base de données.

Question : *Combien il y a de pages indexées ?*

Question : *Combien de pages ont la même URL requêtée et répondue ?*

Question : *Certaines pages comme `https://wiki.jachiet.com/wikipedia_fr_mathematics_nopic_2020-04/A/Surface_de_Delaunay` ne sont pas indexées, pourquoi ? Pouvez-vous en trouver d'autres ?*

2 Deuxième phase : précalcul de TF et IDF

Le rendu de cette partie doit être dans le fichier `index.py`. Attention, il est important que votre indexation soit faite entièrement par le fichier `index.py` (création des tables inclue). Il faut que je puisse simplement lancer la commande `python3 index.py` avec la base de données que je vous ai fournie.

Si vous voulez faire des tests je vous conseille de créer la table avec les commandes suivantes (la première supprime l'éventuelle ancienne et la seconde créé une nouvelle) :

```
conn.execute("DROP TABLE IF EXISTS inverted_index")
conn.execute("CREATE TABLE inverted_index (columnA TEXT, columnB REAL)")
```

De la même façon si vous créez des index, ils seront supprimés automatiquement à la suppression de la table correspondante. Vous pouvez les créer avec :

```
conn.execute("CREATE INDEX name_of_the_index ON inverted_index (columnA)")
```

Pour réduire un peu la taille du dictionnaire, il vous est recommandé d'utiliser une fonction de stemming, même minimaliste comme celle fournie dans `shared.py`.

2.1 Sous-phase 1 : calcul de l'index inversé

Dans cette partie on veut calculer un index inversé. Pour rappel un index inversé c'est quelque chose qui associe à chaque mot clef w une liste $(u_1, f_1) \dots (u_k, f_k)$ où les u_i sont des URL de documents qui contiennent le mot w et f_i est la fréquence d'occurrence de w dans u_i.

L'index sera stocké dans la base SQLite avec une table contenant trois colonnes "keyword", "URL" et "frequency". "keyword" correspond à un mot dans le document à l'adresse "URL" et "frequency" correspond à la fréquence d'apparition du mot dans le document. Attention la fonction de stemming utilisée ici doit être la même que celle pour le requêtage.

Rappel : ici tf correspond à la fréquence d'apparition d'un mot dans une page (donc nombre d'occurrences du mot dans le page divisé par nombre de mots dans la page).

Étape 1 : faire une fonction `countFreq` qui prend une liste L de mots et calcule la liste $(w_1, f_1) \dots (w_k, f_k)$ où w_i est un mot de L et f_i est sa fréquence d'apparition.

Étape 2 : faire une boucle qui itère sur toutes les pages de la table `webpages`. Pour chaque page, elle extrait la liste des mots avec `extractListOfWords`, puis elle appelle la fonction `countFreq` puis elle stocke le résultat dans une table SQL.

Faire le calcul de l'index inversé.

Question : *Dans quelle page apparaît le plus souvent (en fréquence) le terme "matrice" ?*

2.2 Sous-phase 2 : précalcul de l'inverse document frequency

Votre algorithme d'indexation devra aussi gérer le calcul de l'Inverse Document Frequency qui devra aussi être stocké dans une table.

Rappel, l'IDF est ici calculé comme $log\left(\frac{N}{n_w}\right)$ où N est le nombre de documents et n_w le nombre de document où w apparaît. Ici log est entendu comme le logarithme naturel (en base e) qui se trouve en python dans `math.log`.

Question : *Quel est l'IDF de "matrice" ?*

3 Calcul du PageRank

Pour cette partie il faut remplir le fichier `pagerank.py` de sorte que ce programme calcule le PageRank de notre corpus. Comme on ne veut pas recalculer le graphe à chaque itération, on va d'abord calculer ce graphe une fois pour toutes les itérations et ensuite on fera le PageRank sur ce graphe.

3.1 Calcul du graphe

Dans cette première sous-section on veut matéraliser le graphe sur lequel on va ensuite faire le PageRank. Ce graphe on peut le stocker comme un `dict()` dont les clefs sont des respURL et les valeurs sont des listes de respURL.

Dans chaque page web P que l'on a stockée à l'étape du crawling, on trouve des liens (sous la forme de `<a href="url du lien">`). Si on récupère ces liens ça donne une liste d'URL $u_1 \dots u_k$ vers lesquelles P pointe. Cependant les URLs $u_1, \dots u_k$ sont des URL de requêtes, il faut donc aller chercher dans la table `responses` l'URL de réponse correspondantes.

Étape 1 : calculer un dictionnaire `realURL` qui retourne la `respURL` associée à une `queryURL`.

Étape 2 : calculer un dictionnaire `pointsTo` qui stocke la liste `neighbors(content)` pour chaque `respURL`.

Étape 3 : Modifier `pointsTo` pour que les `queryURL` soient remplacés par des `respURL`.

Dans le fichier `shared.py` vous trouverez une fonction qui extrait les liens d'un document. Attention cette fonction renvoie les URL qui apparaissent dans les documents modifiées pour les rendre globales mais ces URL sont celles que notre crawler a demandées et donc éventuellement différentes de ce que le serveur a répondu. Il va donc falloir utiliser la table "responses" pour avoir les bonnes URL.

3.2 Calcul du graphe

Pour le PageRank, on va utiliser $\alpha = 0.15$ (un "damping factor" de 0.85) c'est à dire qu'un marcheur aléatoire a 0.15 chances de se téléporter et ne faire que 50 itérations.

Attention la multiplication de matrices est un algorithme trop lent ici (même avec l'exponentiation rapide). Voici un pseudo code pour le calcul du PageRank :

```
score = [ 1/n pour chaque page]

pour chaque itération:
|  nouveau_score = [ 0 pour chaque page ]
|  proba_téléportation = 0
|  // traitons les sauts de page en page
|  pour chaque page P:
|  |
```

```
|  |   s'il y a des liens sur P:
|  |   |  proba_téléportation += ALPHA × score[P]
|  |   |  pour chaque lien de P vers P' :
|  |   |  |  nouveau_score[P'] += score[P] × (1−ALPHA) / nombre de liens sur P
|  |   sinon (s'il n'y a pas de lien sur P):
|  |   |  |  proba_téléportation += score[P]
|  |   +
|  +
|  // traitons les téléportations
|  pour chaque page P:
|  |  nouveau_score[P] += proba_téléportation / nombre de pages
|  +
|   score = nouveau_score
+
```

Question : *Quelles sont les 20 pages qui ont le meilleur PageRank ? Comment l'expliquez-vous ?*

4 Requêtage

Dans cette partie il faut compléter `query.py`. Quand on lance `query.py` sans arguments il doit lire une phrase sur l'entrée standard (avec `input()`) et fournir une liste ordonnée (du meilleur au moins bon) des 10 meilleurs résultats (avec éventuellement moins 10 si un des mots n'apparaît pas) selon les métriques suivantes.

4.1 Avec simplement tf-idf

Implémenter le tri en utilisant uniquement la métrique tf-idf, c'est à dire selon $\sum \text{tf}(\text{term}, \text{doc}) \times \text{idf}(\text{term})$.
Question : *Quelles sont les dix premières pages pour "comment multiplier des matrices" ?*

4.2 Avec tf-idf et PageRank

Implémenter le tri en utilisant tf-idf multiplié par le PageRank de la page.
Question : *Quelles sont les dix premières pages pour "comment multiplier des matrices" ?*