RAPPORT DU PROJET

Sara Bahadori 22115157

Sonia Dahmani 21965528

TP1 : Corpus

EX1 : explication du code

. Lecture et analyse du fichier XML :

- Le langage choisi pour cette partie est GO pour sa rapidité et flexibilité dans le traitement des larges fichiers XML.

- On a défini une structure 'Page' pour représenter les pages extraites du fichier XML avec des champs 'Title', 'Text' et 'ID'.

- On a créé un sous-ensemble de pages pertinentes en sélectionnant les pages en fonction du mot clé 'histoire' et un nombre de mots minimum.

- Le texte de chaque page sélectionnée a été nettoyé pour éliminer les éléments indésirables et inutile.

- On a assigné à chaque page un identifiant puis remplacé chaque lien interne dans le texte par l'identifiant de la page correspondante pour pouvoir etre récupéré et utilisé dans le calcul du PageRank.

EX2 : explication du code

Le script fourni utilise la bibliothèque `nltk` pour le traitement du langage naturel afin de travailler sur un fichier CSV contenant des pages filtrées. Voici un aperçu du fonctionnement du code :

1. Import des bibliothèques et téléchargement des ressources linguistiques :

Le script commence par importer les bibliothèques nécessaires telles que `nltk`, `Counter` et `pickle`. Ensuite, il télécharge les ressources linguistiques nécessaires de NLTK, telles que les tokenizer et les stopwords pour la langue française.

2. Définition des fonctions auxiliaires :

- `remove_stop_words`: Cette fonction prend une liste de mots en entrée et retourne une liste sans les mots vides (stop words) de la langue française.

- `lemmatize_words`: Cette fonction prend une liste de mots en entrée et retourne une liste de ces mots lemmatisés, c'est-à-dire réduits à leur forme de base.

- `get_top_words`: Cette fonction prend une liste de mots et un entier `n` en entrée et retourne les `n` mots les plus fréquents avec leurs fréquences.

- `print_top_words`: Cette fonction prend une liste de tuples (mot, fréquence) en entrée et les affiche.

3. Fonction principale `main()` :

- Un objet `Counter` est initialisé pour stocker les fréquences des mots.

- Le fichier CSV est ouvert et parcouru ligne par ligne. Pour chaque ligne, le texte de la troisième colonne est extrait.

- Le texte est tokenisé en mots, puis les mots vides sont enlevés et les mots sont lemmatisés.

- Les fréquences des mots sont mises à jour à l'aide de l'objet `Counter`.

- Les 20 000 mots les plus fréquents sont extraits et affichés.

4. Sauvegarde des résultats :

- Les 20 000 mots les plus fréquents sont enregistrés dans un fichier binaire à l'aide de la bibliothèque `pickle`.

En résumé, ce script extrait et analyse les données textuelles d'un fichier CSV, en extrait les mots les plus fréquents après lemmatisation et suppression des mots vides, puis sauvegarde les résultats pour une utilisation ultérieure.

EX4,

Q1: la matrice d'adjacence d'un graphe orienté :

[[0,1,0,0],

[1, 0 ,1,1],

[0,0 ,0,1],

[0,0,0,0]]

Q2 :

Lors de l'indexation de n pages et de la construction d'un graphique des pages visitées, chaque page peut être considérée comme un sommet du graphique. Si nous désignons le nombre de pages par n , alors le nombre de sommets dans le graphique serait également n.

Chaque page correspond à un sommet du graphique, et lorsqu'une page est liée à une autre page, elle crée une arête entre les sommets correspondants du graphique. Cependant, le nombre de sommets reste le même que le nombre de pages indexées.

la taille de la matrice d'adjacence: n*n parce que nous avons n nœuds qui ont une relation avec les mêmes n nœuds

Pour n = 10^9, peut-on stocker une telle matrice en mémoire ? Non, stocker une matrice 10^9×10^9 = 10^18 représentant les connexions entre les pages en mémoire n'est pas réalisable en raison de sa taille énorme parce qu'il est trop volumineux pour être conservé en mémoire.

Q3 :

Si chaque page a 10 liens en moyenne, il y aura en moyenne 10 coefficient non nul pour chaque nœud (page) donc il y aura 10n coeffcients non nuls dans la matrice , donc avec n= 10^9 le nombre de coeffcients non nuls dans la matrice : 10 n = 10^10.

Si on enlève tous les zéros, on peut stocker les coeffcients en mémoire. parce que 10^10 est tellement inférieur à 10^18.

EX5 :

Q1 : C=[1,2,3,4,5 ,5,7]

I=[2,0,1,3,1,2,3]

L=[0,1,4,7,7]

Q2,Q3 : explication du code

Le code fourni comporte plusieurs fonctions qui effectuent des opérations sur une matrice. Voici un résumé des différentes parties du code :

1. Fonction `calculate_result` :

Cette fonction prend une liste triée en entrée et un nombre `nb`. Elle calcule le nombre d'occurrences de chaque élément jusqu'à `nb` et stocke ces valeurs cumulatives dans une liste `result`.

2. Fonction `matrix_to_CLI` :

Cette fonction prend une matrice sparse en entrée et la convertit en trois vecteurs :

- `C` contenant les valeurs non nulles de la matrice.
- `I` contenant les indices des colonnes correspondant à chaque valeur non nulle.
- `L` contenant les indices de début et de fin de chaque ligne dans le vecteur `C`.

3. Exemple de matrice et application de `matrix_to_CLI` :

Une matrice `matrix` est définie comme un exemple. `matrix_to_CLI` est ensuite appliquée à cette matrice pour obtenir les vecteurs `C`, `I` et `L`.

4. Fonction `Prod` :

Cette fonction effectue le produit matrice-vecteur en utilisant les vecteurs `C`, `I` et `L` calculés précédemment. Elle parcourt les lignes de la matrice en utilisant les informations dans le vecteur `L` pour calculer le produit scalaire de chaque ligne avec le vecteur `M`.

5. Exemple d'utilisation de `Prod` :

Un vecteur `M` est défini comme exemple. La fonction `Prod` est appliquée aux vecteurs `C`, `I`, `L` et au vecteur `M` pour obtenir le résultat du produit matrice-vecteur.

En résumé, le code fourni effectue des opérations de manipulation de matrices, telles que la conversion d'une matrice sparse en vecteurs compacts et le calcul du produit matrice-vecteur à l'aide de structures de données optimisées.

EX6 :

Q1 : dans le graphe des liens entre pages, les sommets du graphe représentent les pages du corpus, chaque sommet correspond à une page, un arc (i, j) représente un lien qui va de la page i vers la j

Q2 : avec *n* pages visitées, et si chaque page contient en moyenne 500 mots *différents* du dictionnaire 500n éléments la relation mots-pages va contenir.

Ce nombre ne dépend pas de *m et* il raisonnable de vouloir indexer toutes les pages du corpus parce que ça contiendrait trop de zéros inutiles.

EX7 :

Q5 : si on avait pas le fichier à disposition, il faudrait collecter les pages nous-meme, on peut commencer par la page d'acceuil de wikipedia puis parcourir de là les liens sortants de cette page, les suivre et ainsi découvrir de nouvelles pages et ainsi de suite.

EX3, EX7, EX8 : explication des codes

Le script commence par importer les bibliothèques nécessaires telles que `math`, `csv`, `nltk` et `json`. Ensuite, il télécharge les ressources linguistiques nécessaires de NLTK, telles que les tokenizer et les stopwords pour la langue française.

Ensuite, deux fonctions sont définies :

1. `remove_stop_words`: Cette fonction prend une liste de mots en entrée et retourne une liste sans les mots vides (stop words) de la langue française.

2. `lemmatize_words`: Cette fonction prend une liste de mots en entrée et retourne une liste de ces mots lemmatisés, c'est-à-dire réduits à leur forme de base.

La fonction principale `main()` est définie ensuite. Dans cette fonction :

- Un dictionnaire `word_freq` est initialisé pour stocker les fréquences des mots.

- Le fichier CSV est ouvert et parcouru ligne par ligne. Pour chaque ligne, le texte de la troisième colonne est extrait.

- Le texte est tokenisé en mots, puis les mots vides sont enlevés et les mots sont lemmatisés.

- Les fréquences des mots sont mises à jour pour chaque page.

- Le nombre total de documents est calculé.

- Seuls les 20 000 mots les plus fréquents sont conservés.

Après cela, deux autres fonctions sont définies :

1. `calculate_idf_and_tf`: Cette fonction calcule l'IDF (Inverse Document Frequency) et le TF (Term Frequency) pour chaque mot et met à jour le dictionnaire `word_freq`.

2. `calculate_Nd`: Cette fonction calcule la norme des vecteurs des fréquences des termes (TF) pour chaque document.

Ensuite, la fonction `update_word_freq` est définie pour mettre à jour les valeurs TF-IDF pour chaque mot dans chaque page.

Enfin, une fonction `save_dict_to_json_stream` est définie pour sauvegarder le dictionnaire final contenant les valeurs TF-IDF dans un fichier JSON.

Le code principal utilise ces fonctions pour traiter les données, calculer les valeurs TF-IDF et les sauvegarder dans un fichier JSON.

TP2,

Matrice CLI:

1. On parcourt, à l'aide d'une fonction, le fichier CSV qui représente notre corpus, afin d'extraire tous les liens internes de chaque page puis générer une représentation CLI de la matrice d'adjacence du graphe.

- Une fois la matrice générée, on sauvegarde le résultat dans un fichier JSON pour éviter de recalculer la matrice.

2. Transposée de la matrice creuse : La fonction 'transpose_stochastiquement' calcule le produit matriceXvecteur sans calculer explicitement la transposée de la matrice creuse 'Ag'. Le calcul se fait en un seul parcours des tableaux 'C', 'L', 'I' avec une complexité O(n+m).

Une variante est proposée ci-dessus n'utilisant pas de tableau C étant donné que l'on peut facilement déduire les coefficients grâce aux valeurs de L et I:

```python
def transpose_stochastiquement(L, I, V):
    n = len(L) - 1  # nombre de ligne de la matrice
    P = np.zeros(n)  # initialisation avec des zéros

    for i in range(n):
        if L[i] == L[i + 1]:  # ligne vide
            for j in range(n):
                P[j] += V[i] / n
        else:
            for j in range(L[i], L[i + 1]):
                P[I[j]] += V[i] / (L[i + 1] - L[i])
    return P

```

TP3,

Pagerank:

- Afin d'évaluer l'importance et la pertinence des pages Wikipedia en fonction des liens du graphe, nous avons utilisé l'algorithme du PageRank.

- Nous avons appliqué l'algorithme du PageRank aux données collectées lors du TP 1 en utilisant la fonction generate_pageRank. Cette fonction génère la matrice CLI à partir des données des pages collectées, initialise la distribution initiale Pi0 uniformément sur les n sommets, puis calcule le PageRank après un nombre d'itérations k qui a été choisi en fonction de la taille et du temps que prenaient les calculs.

- Le paramètre 'epsilon' représente la probabilité de 'saut' aléatoire d'une page à une autre, afin de simuler au mieux le comportement d'un vrai utilisateur naviguant sur le web. On a choisi la valeur de 1/7 qui est un choix courant et une assez petite valeur pour converger rapidement. Nous avons expérimenté avec d'autres valeurs plus grandes ou plus petites mais 1/7 donnait les résultats les plus constants.

Scores de fréquence :

- Nous avons calculé le score de fréquence pour chaque page du corpus par rapport à une liste de mots clés de la requête.

- La fonction 'frequence_score' calcule les scores de fréquence des pages par rapport à une requête en combinant les scores TF-IDF de chaque mot de la requête avec les scores de PageRank des pages calculés précédemment.

- On ne parcours qu'une fois la liste des mots de la requête, en récupérant d'abord toutes les pages contenant chaque mot de r, puis en faisant l'intersection de ces ensembles pour obtenir les pages qui contiennent tous les mots.

- Après calcul, les résultats sont triés par ordre décroissant afin de classer et de présenter les résultats de manière pertinente pour l'utilisateur.

- choix des paramètres alpha et beta: l'utilisation des deux paramètres a pour but d'équilibrer le poids du PageRank par rapport à la fréquence des mots dans le calcul du score final:

- On choisit de prendre alpha=1, puis sélectionner Beta comme étant m1//m2 où m1 est la moyenne des scores de fréquence des pages et m2 est égal à 1/n où n est le nombre total de pages.

- Pour obtenir la moyenne des scores de fréquence, on choisit des requêtes aléatoires dont on calcule le score et la moyenne.

- Cela garantit que le pagerank ne sera pas trop négligé par rapport au score de fréquence dans le calcul final du score d'une page par rapport à une requête donnée.

Déploiement :

- Nous avons créé un site basique mais fonctionnel qui permet à l'utilisateur de saisir une requête de recherche puis afficher une liste de résultats.

- Le serveur utilise Flask et a deux routes principales : une route pour afficher la page d'accueil et une autre pour traiter les requêtes de recherche.

Difficultés:

- Nous avons rencontré des difficultés majeures liées à la manipulation de fichiers de très grande taille, entraînant des ralentissements significatifs des calculs, surtout sur des machines moins puissantes. Pour surmonter ces obstacles, des algorithmes performants ont été recherchés et implémentés, avec une attention particulière portée à l'optimisation de l'espace mémoire. Cette optimisation a nécessité l'utilisation de structures de données efficaces et la réduction de la consommation de mémoire.