

Année universitaire 2019-2020

**Méthodes algorithmiques pour l'accès à l'information numérique**

**Projet : Moteur de Recherche**

**Rapport**

**Réalisés par : Professeurs :**

**André KENY 21605009 Pierluigi CRESCENZI**

**Nassim OUHENIA 21703317 Dominique POULALHON**

**Introduction**

Le but de ce projet est de réalisé un moteur de recherche en respectant les cahiers de charge. On est parti d’un gros fichier qu’on a réduit par rapport aux thèmes que l’on a choisi ; ici dans notre cas on parlera de Musique, Cinéma et Artiste. On peut remarquer que les 3 thèmes ont pour idée commune l’art. Comme langage de programmation, on a utilisé du Python, car lors de notre parcours pédagogique, on est amené souvent à utiliser plus le langage Java que les autres, donc cette fois ci, on s’est mis comme challenge d’essayer de faire le projet en Python pour accroitre notre connaissance en Python et aussi changer un peu.

Nous allons d’abord commencer par élaborer notre base, créer notre base des mots les plus fréquents, ensuite générer notre matrice puis ensuite notre fichier collecteur

**Corpus**

**Sélection et nettoyage**

En inspectant le fichier frwiki.xml, on a remarqué que les pages qui parlaient de musique avait pour mot commun « Infobox musique » ; on s’est basé sur ce mot pour avoir les pages qui parlé de musique ainsi que celle parlant de « Cinéma » et « Artiste ». A la fin on s’est retrouvé avec un portefeuille de page avoisinant les 200000 pages.

On a gardé le mot infobox car par exemple pour sélectionner les pages parlant de musique, on aurait pu utiliser des mots spécifiant le domaine musique comme « musique, music, musical, chanson, chanteur, compositeur, … » mais on s’est rendu que l’on collectait des pages ne parlant pas de musique. Par exemple on avait collecté une page parle de l’arc de triomphe. Donc avec infobox musique, on filtrait plus sévèrement les pages que l’on voulait sélectionner. L’ensemble des pages sélectionnées forment notre fichier wiki\_musique\_artiste\_cinema.xml

On a un fichier corpus.xml qui comporte la classe Corpus ; le constructeur de cette classe prend comme argument :

* Name\_file : le gros fichier xml
* Key\_words : qui est une liste des mots à chercher dans le gros fichier
* Out\_file : le fichier de sortie qui est une réduction du gros fichier donné en entrée

Ensuite on a une fonction principale :

* Reduction : qui ouvre le fichier de sortie en écriture, avec une écriture ligne par ligne. Ouvre le gros fichier en lecture en parcourant page par page. Le choix d’écrire ligne par ligne et de parcourir page par page est par soucis de performance car c’est mieux optimiser et on ne risque pas de saturer la mémoire de l’ordinateur. Du coup on est sûr de la lecture d’un gros fichier et de son écriture aussi. A la fin on a notre fichier qui fait soit 200.000 pages ou un peu moins.

**Dictionnaire**

Notre dictionnaire est composé de 10.000 mots triés par ordre décroissant d’occurrence. Pour prendre que les mots importants, on a dû supprimer les mots vides en utilisant le module stopword en python et ajouter dans ce stopword des mots inutiles échappant aux filtrages stopword. Car lors du formatage du texte, on a eu des mots bizarres qui ne signifiaient rien, donc a décidé de faire une liste que l’on met régulièrement à jour lors de l’inspection des 10000 mots. Et la liste se nomme listeMotAEnlever.

C’est ainsi que l’on a notre Classe Dictionnaire qui prends en entrée notre fichier corpus et une liste des mots à enlever. On a combiné deux procédés dans cette classe :

* Le fait d’avoir la liste des 10.000 mots les plus fréquents
* Le fait d’avoir la liste des pages auxquels est présent un mot

Car cela nous permet de parcourir notre fichier corpus une fois pour des soucis de performance.

Vu que l’opération d’avoir pour chacun des 10.000 mots, la liste des pages contenant ce mots prends du temps, on a pris l’initiative de sauvegarder le résultat dans un fichier nommé RelationMotPage.txt où on a une représentation suivante mot;page1;page;page3 ;…

**Matrice**

**Exo3 : Graphe**

1) Matrice d’adjacence 0 1 0 1

0 0 1 0

0 0 0 1

1 1 1 0

2) Si on indexe n pages, le nombre de sommets du graphe des pages visitées k est : k<= n-1

La taille de la matrice d’adjacence est : n\*n

Pour n= 109, on ne peut pas stocker une telle matrice en mémoire car la taille de la matrice sera de 1018 et ceci ne pourra pas rentrer en mémoire

3) Si chaque page a 10 liens en moyenne, le nombre de coefficients non nuls dans la matrice précédentes et de 109\*10 donc 1010

Si on enlève tous les zéros, il est possible d’enregistrer les coefficients en mémoire.

Exo4 Matrices

Collecteur

Exercice 5 : Taille des structures

1)Les sommets du graphe sont les titres des pages

Les arcs sont les coefficients vers les liens externes

Le nombre de sommets de notre graphe : 191187

Le nombre d’arrête de notre graphe : 2028758

La taille du codage CLI :

C = nombre d’arrêtes C = 2028758

L = nombre de sommets +1 donc L= 191188

I= nombre d’arrêtes 2028758

2)le nombre d’éléments que va avoir la relation mot-pages est de : 200\*m\*n

Ce nombre