Méthodes algorithmiques pour l'accès à l'information numérique

Gougeulle, le moteur de recherche de l'avenir

Université de Paris

Aliaksandr Budzko,
Chawki Chouib,
Benjamin Viau,

21 mars 2021

Table des matières

1 Introduction

Mes chers camarades et moi-même eûmes la chance de travailler sur un moteur de recherche pour le Wikipédia français que nous allons vous détailler prestement.

2 Implémentation

Dans cette partie on explique nos choix pour extraire le corpus de travail du dump Wikipédia donné.

2.1 Préparation du corpus

Nous avons eu à notre disposition près de 5 millions de pages Wikipédia sous forme d'un fichier de 25go. On avait comme consigne de travailler sur une partie de cet ensemble de pages uniquement, notamment en choisissant un thème.

Pour procéder, nous avons choisi de se limiter aux pages Wikipédia ayant pour sujet la littérature.

Pour commencer , nous avons parsé le grand fichier XML, tout en faisant attention à ne pas le mettre entièrement en mémoire, afin d'en tirer seulement les informations dont on aurait besoin par la suite.

```xml
<mediawiki>
    <siteinfo>
        <sitename>...</sitename>
        <dbname>...</dbname>
        <base>...</base>
        <generator>...</generator>
        <case>...</case>
        <namespaces>
            <namespace>...</namespace>
        </namespaces>
    </siteinfo>
    <page>
        <title>...</title>
        <ns>...</ns>
        <id>...</id>
        <revision>
            <id>...</id>
            <parentid>...</parentid>
            <timestamp>...</timestamp>
            <contributor>
                <username>...</username>
                <id>...</id>
            </contributor>
            <minor/>
            <model>...</model>
            <format>...</format>
            <text>...</text>
        </page>
</mediawiki>
```

Listing 1 – Structure XML avant

Ainsi, en enlevant toutes les balises XML non nécessaires et en filtrant les pages en fonction de leur catégorie, on obtient un nouveau fichier XML qui contient seulement les pages appartenant au domaine de la littérature.

```xml
<mediawiki>
    <page>
        <title>page 1</title>
        <id>1</id>
        <text>...</text>
        </page>
    <page>
        <title>page 2</title>
        <id>2</id>
        <text>...</text>
    </page>
</mediawiki>
```

Listing 2 – Structure XML après

De plus, en même temps que l'on écrivait dans ce fichier, on remplissait une liste **pagelist** ayant la forme $(id, titre, contenu)$ et qui a par la suite été sérialisée et sauvegardée sur le disque.

2.2 Graphe

Tout d'abord, nous faisons un premier parcours de la liste de pages pour créer une structure de données contenant toutes les arêtes du graphe (les liens des pages). Puis nous créons simplement le graphe en forme CLI, en prenant soin de ne pas prendre en comptes les arêtes qui pointent en dehors de notre corpus.

2.3 Nettoyage du corpus

Cette partie est consacrée à expliquer comment on a implémenté le passage d'un contenu de départ (avec des éléments de mise en page etc) à un contenu propre et prêt à être parsé.

FIGURE 1 – Preview structure de départ

Dans la figure ci-dessus nous pouvons voir un exemple de structure de page dans le fichier wiki.xml.

FIGURE 2 – Preview structure après nettoyage des balises XML

Dans la figure ci-dessus nous pouvons voir la page précédente nettoyé de ses balises inutiles pour notre projet.

En effet nous avons gardé que les informations importantes pour notre projet, c'est a dire l'identifiant de la page, son titre et son contenu.

Pour ce qui est du contenu de la page, ce dernier va être nettoyé pour le transformer en texte clair (voir la figure suivante).

FIGURE 3 – Preview structure après nettoyage du text

Dans la figure ci-dessus nous pouvons voir le résultat du nettoyage de la page Wikipédia pour la rendre lisible.

2.3.1 Nettoyage

Le processus de nettoyage prend en entrée la liste des pages et modifie leur contenu de manière suivante :

- suppression des balises "internes" (<>, []...)
- suppression des URL
- suppression des caractères de mise en page (par exemple le """ au début de l'article)
- suppression des sections non pertinentes (par exemple "=== Références ===")
- suppression des espaces blancs
- **tokenisation et lemmatisation** avec Spacy (voir plus bas)
- suppression de la ponctuation
- sauvegarde de la version nettoyée en lettres minuscules

La nouvelle version de la **pagelist** est également sérialisée et stockée sur le disque.

2.3.2 Tokenisation

Pour tokeniser le contenu d'une page, nous avons choisi d'utiliser la bibliothèque Spacy, car malheureusement nltk, étant très bien pour du traitement de texte en anglais, s'est avéré moins efficace en français. Nous appliquons la tokénisation avant de mettre le texte en minuscule et avant d'enlever la ponctuation car Spacy est capable de comprendre le contexte des mots dans une phrase et, grâce à cela, les lemmatiser de manière plus précise.

2.4 Dictionnaire

Pour le dictionnaire nous avons choisi la structure suivante :

```
{mot:({page:freq},-)}
```

Pour à la fin retourner :

```
{mot:({page:TF_norm},IDF_mot)}
```

Nous initialisons 2 dictionnaires, un **dico-titre** pour les mots des titres et un deuxième **dico-text** pour les mots des textes.

En parcourant chaque page on remplit **dico-text** en incrémentant de 1 la fréquence de chaque mot qu'on rencontre et en ajoutant de nouveaux mots si besoin. On fait de même avec **dico-titre** sauf qu'on incrémente de 100 la fréquence pour donner plus d'importance aux mots des titres.

Nous fusionnons ensuite **dico-titre** avec les 200000 mots les plus fréquents du **dico-text** en obtenons de cette manière notre dictionnaire.

Nous calculons et normalisons ensuite le TF de chaque mot pour chaque page, puis nous calculons le IDF de chaque mot.

Ainsi, nous obtenons notre structure de retour.

2.5 Pagerank

À partir de notre matrice CLI, nous faisons d'une pierre deux coups en calculant le produit $M^t * R$ directement, c'est-à-dire sans passer par le calcul de M^t.

Nous avons choisi un paramètre optionnel pour initialiser le nombre d'itération du produit matriciel, et ainsi contrôler au mieux la précision des probabilités.

Pour le choix de nos α et β, nous avons comparé les ordres de grandeur des pagerank et des $f(d, r)$ sur un ensemble de requêtes aléatoires et avons pris la différence moyenne.

2.6 Requête

La requête de l'utilisateur consiste en un ensemble de mots. Elle est nettoyée par la même fonction qui nettoie le texte des pages pour au final avoir un ensemble de mots compris dans notre dictionnaire de relation mots-pages.

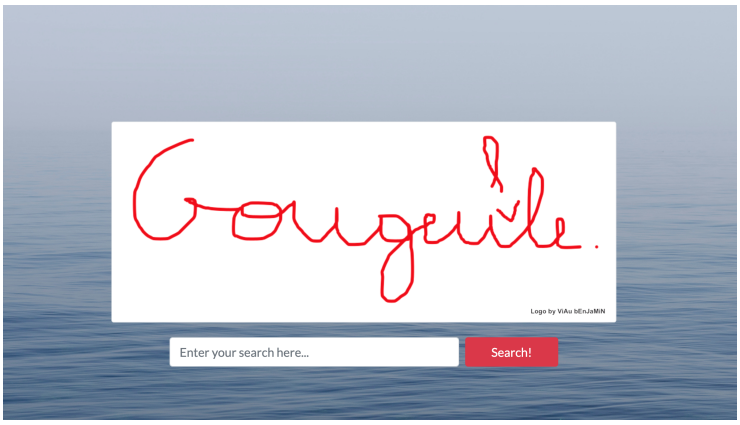
Une fois notre ensemble de mots de la requête prêt, nous utilisons notre dictionnaire de relation mots-pages pour énumérer toutes les pages contenants nos mots de la requêtes, puis on calcule le score de chacune des pages pour les afficher dans l'ordre décroissant.

2.7 Serveur

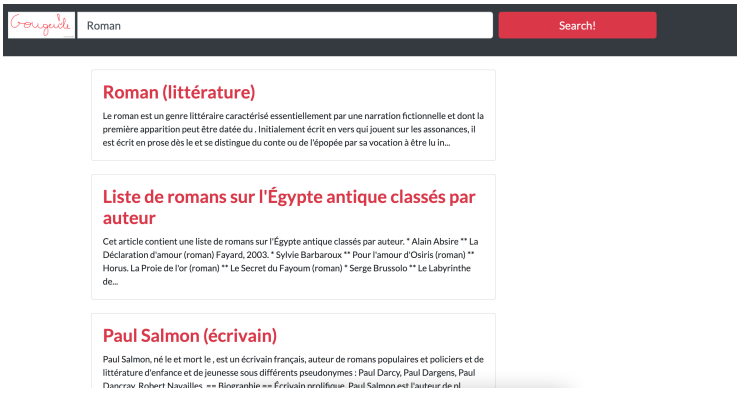
Pour le serveur Web, nous avons décidé d'utiliser Tornado. C'est un framework assez minimaliste pour créer des applications Web interactives de manière intuitive et rapide.

Du côté de l'interface utilisateur, le serveur imite les moteurs de recherche modernes avec une page d'accueil classique et une page d'affichage des résultats. (voir les figures ci-dessous)

Le serveur desérialise les structures nécessaires au démarrage et ensuite attend de recevoir des requêtes utilisateur.



Page d'accueil



Page des résultats de la recherche pour romans

2.8 Résumé

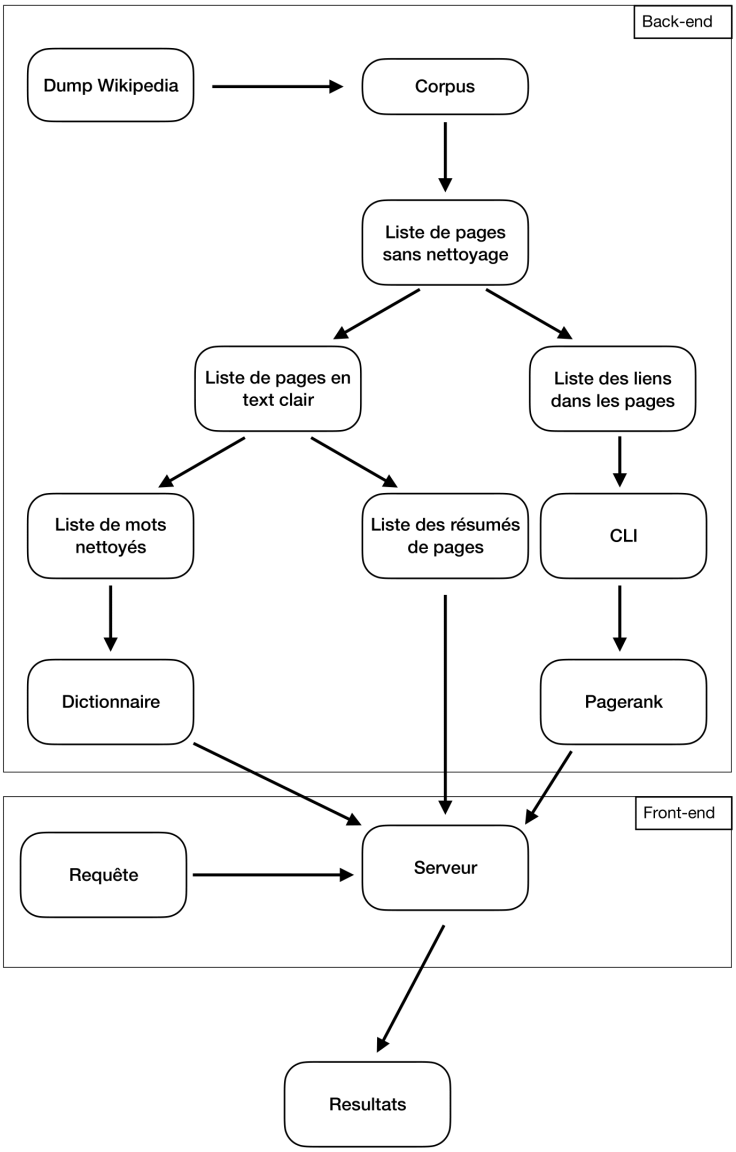


Schéma du projet

3 Réponses aux TP's

Nous avons déjà répondu à la plupart de questions dans les sections précédentes, cette partie contient celles qu'on n'a pas pu y insérer.

3.1 TP1

Exercice 5 Matrices

1)
C = [1,2,3,4,5,6,7]
L = [0,1,3,7,7]
I = [2,0,1,3,1,2,3]

2) Voir le code source.

Exercice 6 Taille des structures

1) Les sommets correspondent aux pages du corpus et les arêtes aux liens. On a 252375 sommets et 2958626 liens.
len(C) = 2958626
len(L) = 252375
len(I) = 2958626

2) On aurait au plus 200*n éléments dans la relation mots-pages. Ce nombre ne dépend pas de m.

Exercice 7 Exploration

5) Si le dump n'était pas disponible, on aurait été obligé de créer un Web crawler. C'est un programme qui parcourirait les pages selon une certaine prioritée en se connectant aux serveurs Wikipédia. Le web crawler devrait également respecter les délais entre deux connexions au même serveur pour ne pas être pris pour une attaque DoS et bani d'accès futurs.

3.2 TP2

Voir le code source.

3.3 TP3

Nous avons choisi la méthode "simple" car nous n'avons pas vraiment eu le temps de bien implémenter la méthode WAND.

Exercice 6 Améliorations possibles

— Implémentation de l'algorithme WAND.
— Rajouter l'index positionnel des mots dans les pages afin d'améliorer la précision des résultats.
— Pré-calculer les résultats des requêtes les plus fréquentes.
— Se servir du multi-threading dans nos pré-calculs pour diminuer la complexité en temps.
— Trouver un moyen de parcourir la pagelist sans la charger entièrement en mémoire, notamment avec méthodes de flots d'entrée.