SORBONNE UNIVERSITÉ

UE DAAR

COMPTE RENDU DU PROJET 4
PROJET FINAL

CHOIX A. MOTEUR DE RECHERCHE D'UNE BIBLIOTHEQUE

06 Février 2022

Réalisé par :

Maedeh DAEIMI

Kaoutar NHAILA

Les enseignants :

Binh-Minh Bui-Xuan

Arthur Escriou

Année universitaire : 2021-2022

Table des matières

1 Introduction

Les bibliothèques électroniques continue de mettre à la disposition de tous la richesse de ses collections et la diversité de son offre numérique. Livres, documents... Des millions de ressources sont accessibles en ligne pour travailler, apprendre ou se détendre. Par contre l'internaute peut être perdu dans ces immenses bibliothèques électroniques.

Dans notre projet, nous avons réalisé une application web de moteur de recherche de document dans une bibliothèque de livres sous format textuels. En autres mots ce projet est pour objectif la recherche des livres dans la bibliothèque à l'aide d'un mot clé ou bien par RegEx.

Notre projet offre aussi deux fonctionnalités implicites, la première fonctionnalité est le classement et la deuxième fonctionnalité c'est la fonctionnalité de suggestion. Plusieurs stratégies nous ont été proposées. Nous avons choisi d'implémenter :le nombre d'occurrences par mot-clé, l'agorithme de Jaccard ainsi que le "closness centrality ranking".

2 Architecture de projet

Notre application se constitue du côté client réalisé en ReactJS et côté serveur réalisé en Spring Boot.

L'application web se constitue de 3 vues : la vue **"Acceuil"** qui contient tous les livres dans notre bibliothèque, la vue **"Recherche Classique"** qui permet à un internaute de faire une recherche classique et la vue **"Recherche Avancée"** qui permet à un internaute de faire une recherche à l'aide d'une expression régulière.

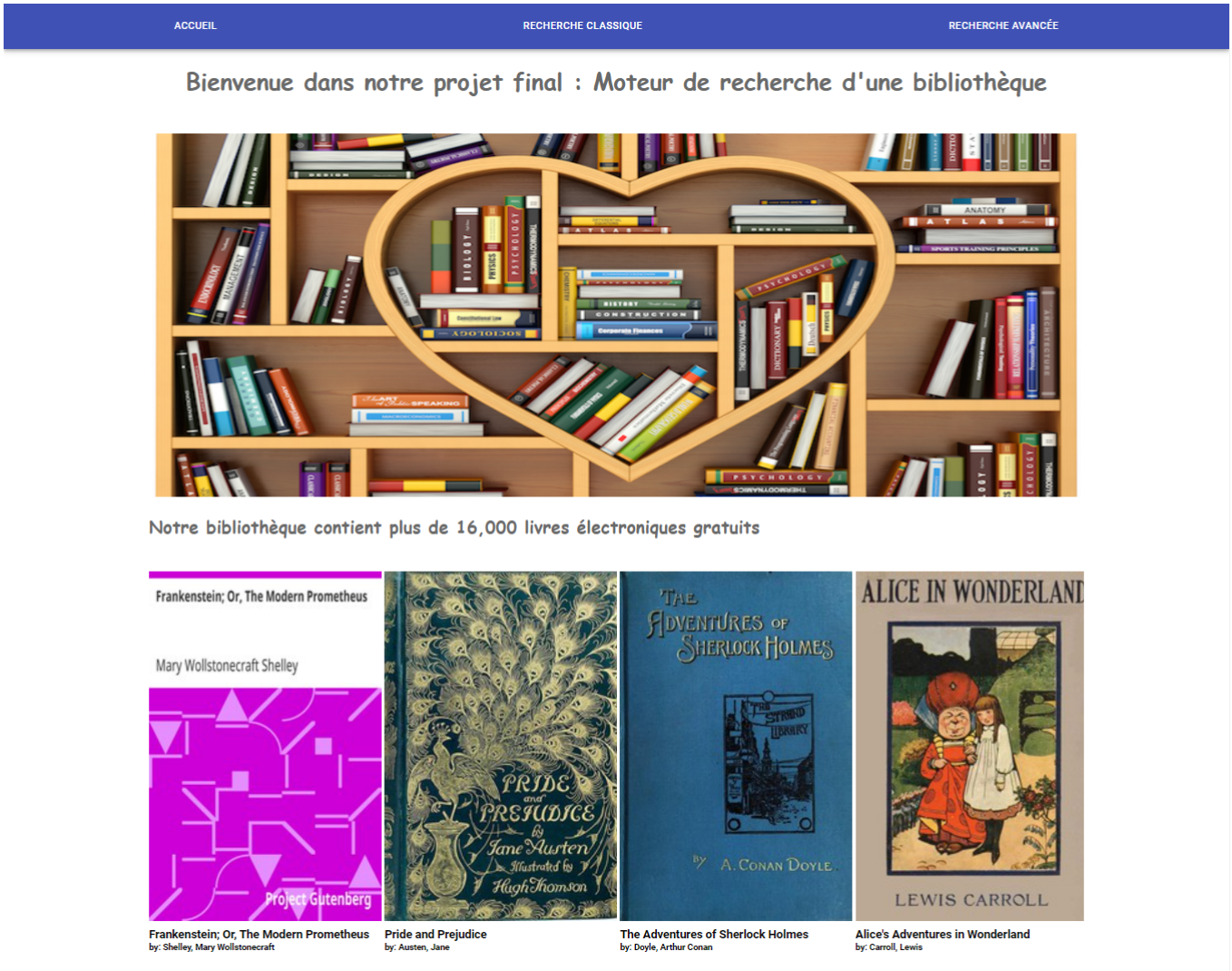


FIGURE 1 – Une vue globale de notre application

Les données telles que les livres, la matrice de Jaccard ou encore le graphe de Jaccard, sont représentées sous la forme de fichiers textuels dans le côté serveur.

Pour chaque livre, on a fichier qui textuel de livre "id_livre.txt", un fichier d'indexage qui contient tous les mots et le nombre d'occurrences de chaque mot "id_livre.dex", un fichier d'indexage sérialisé "id_livre.map".

La figure ci-dessous présente une vision détaillé sur la structure de notre projet :

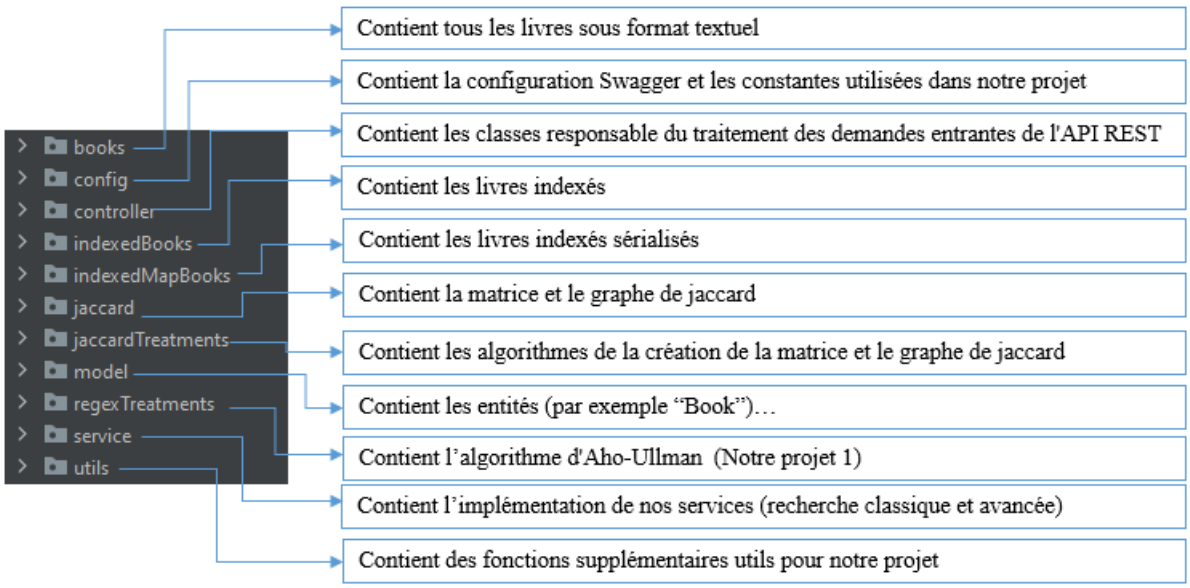


FIGURE 2 – La structure de projet de côté serveur

3 Les fonctionnalités principales de l'application

Dans notre application, on propose de fonctionnalités de recherche principales. Une recherche classique par un mot clé saisit par l'utilisateur et une recherche avancée à l'aide d'une expression régulière.

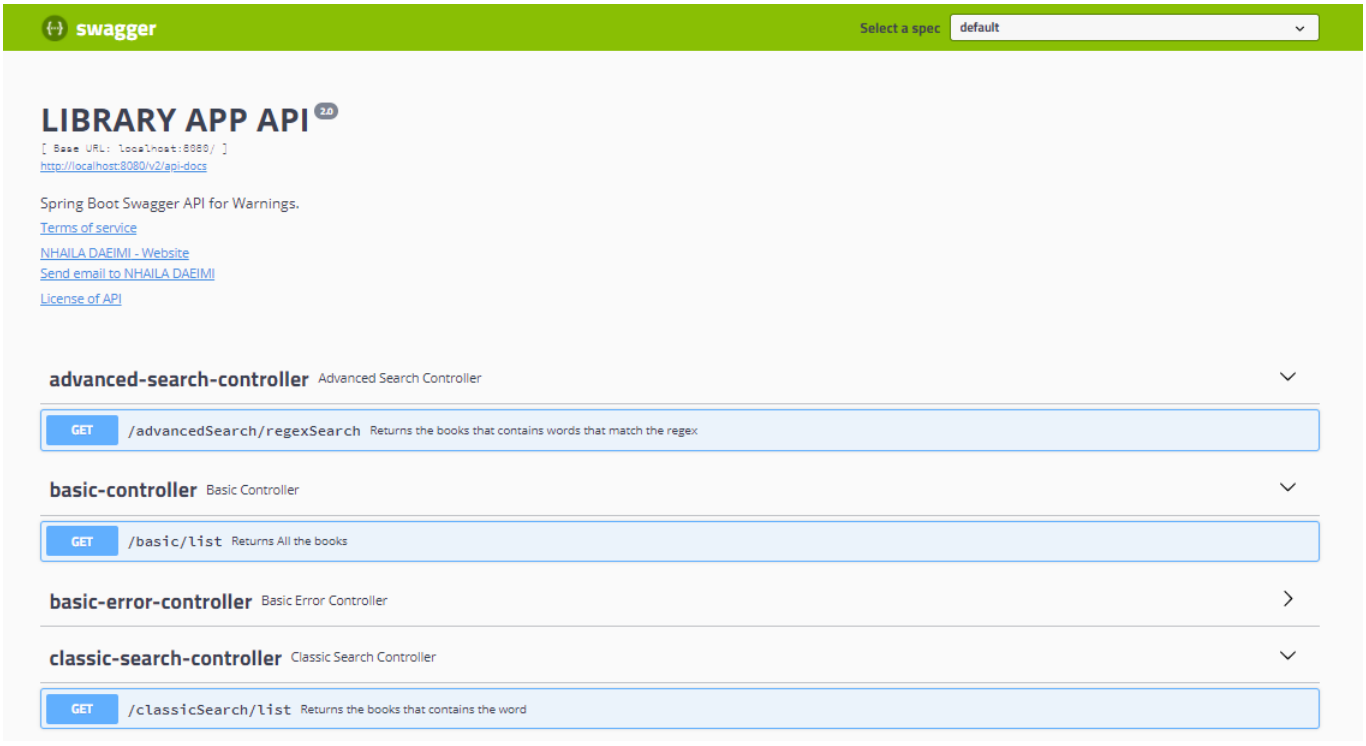


FIGURE 3 – Les endpoints de l'application

3.1 La recherche classique

Cette fonctionnalité permet aux utilisateurs de chercher des livres dans notre base de données à l'aide d'une chaîne de caractère.

Pour réaliser cela, nous parcourons l'ensemble des tables d'indexages de chaque livre présent dans notre base de données. A partir de cette table, nous récupérons à la fois l'identifiant du document et l'occurrence du mot. Ainsi nous obtenons une table hachée avec comme clé : l'identifiant du document et comme valeur : l'occurrence du mot clé. Le nombre d'occurrence permettra de trier cette table. Plus le nombre d'occurrence est élevé, plus le livre est intéressant pour l'utilisateur.

3.1.1 Implémentation

La recherche classique se constitue de plusieurs étapes :

1. Nous récupérons le mot clé saisi par l'utilisateur.
2. Pour chaque livre, nous récupérons sa table d'indexation, si le mot clé existe dans la table, nous ajoutons l'identifiant de ce livre et le nombre d'occurrence trouvé.
3. Nous récupérons les informations des livres dans lesquels le mot clé existe tel que (le titre, l'image de couverture, les auteurs...)
4. Nous trions les livres en fonction de nombre de l'occurrence.

Nous obtenons une complexité en O(n), avec n le nombre de livres car la vérification dans la table d'indexage, crée au préalable, et la récupération de l'occurrence du mot ce fait en O(1).

3.1.2 Amélioration de la recherche

Pour permettre un gain de temps considérable, nous utilisons un framework nommé "Kryo"[7]. Kyro Kryo est un framework de sérialisation de graphes d'objets binaires rapide et efficace pour Java. Les objectifs de ce framework sont une vitesse élevée, une taille réduite et une API facile à utiliser. Le projet est utile chaque fois que des objets doivent être conservés, que ce soit dans un fichier, une base de données ou sur le réseau.

3.2 Recherche Avancée

Cette fonctionnalité permet à l'utilisateur de chercher un livre à l'aide d'une expression régulière qui peut contenir **"*", "|", ".", "concaténation"**.

Cette recherche doit retourner une liste de livre dont la table d'indexage contient une chaîne de caractère vérifiant l'expression. La structure obtenue sera une table hachée avec comme clé les identifiants des livres ayant eu un match avec l'expression régulière, et comme valeur des couples (nombre de match et la somme des occurrences des match). Cette structure sera triée par nombre de match puis par somme des occurrences de manière décroissante.

Notre implémentation de l'algorithme d'Aho-Ullman du premier projet a été réutilisée pour tester l'expression régulière sur chaque mot de la table d'indexage.

3.2.1 Implémentation

L'algorithme de la recherche avancée se décompose en 7 grandes étapes :

1. Nous génèrons un arbre binaire (Regex Tree) à partir de l'expression régulière saisi par l'utilisateur.
2. A partir de l'arbre, nous déterminons l'automate fini non-déterministe normalisé avec epsilons transitions qui accepte le langage dénoté par l'arbre binaire selon la méthode Aho-Ullman.
3. Nous construisons l'automate fini déterministe en supprimant les transitions epsilons grâce à la méthode des sous-ensembles.
4. Nous minimisons l'automate en construisant un automate équivalent mais avec un nombre minimum d'états (Optionnelle).
5. Nous construisons le programme de reconnaissance à partir de l'AFD.
6. Nous récupérons la table d'indexage de chaque livre, on vérifie mot par mot, est-ce qu'il existe une mot vérifiant l'expression régulière.
7. Nous trions les livres dans lesquels existent des mots vérifiant l'expression régulière par nombre de match puis par somme des occurrences de manière décroissante.

3.2.2 Les étapes de l'algorithme Aho-Ullman

Nous allons présenter brièvement l'algoritme d'Aho-Ullman implémenté dans le projet 1 dans l'exemple suivant :

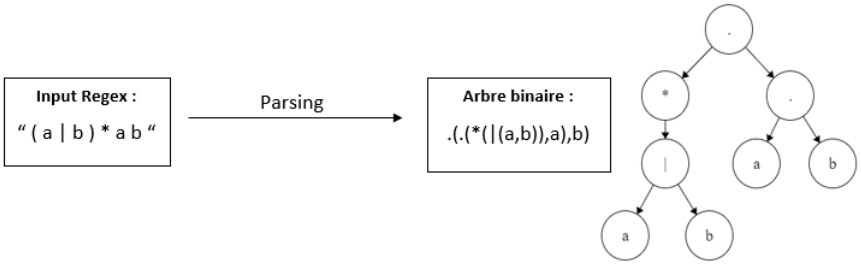


FIGURE 4 – La transformation d'une expression régulière à un arbre binaire (Étape 1)

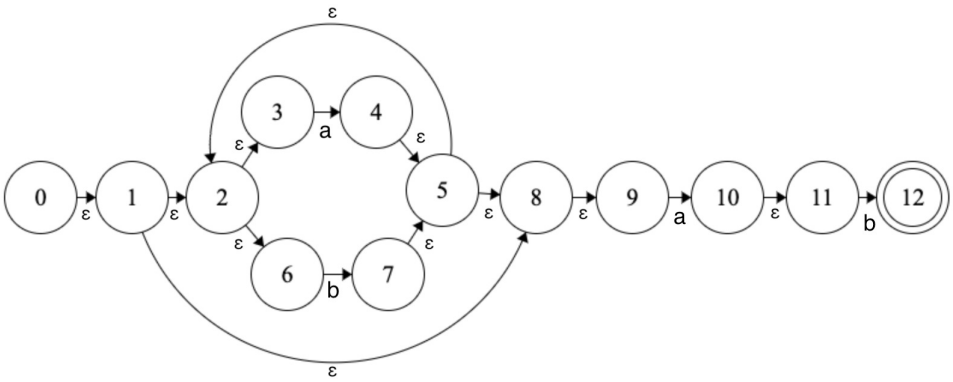
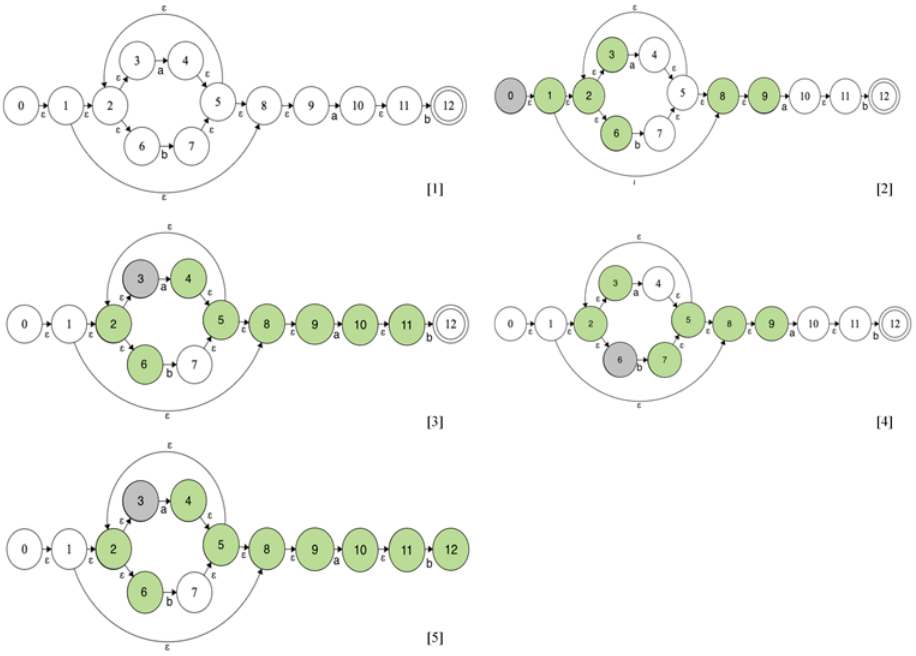


FIGURE 5 – L'Automate Fini Non Déterministe de lexpression régulière **"(a|b)*ab"**

N° du noeud \ ASCII	0	1	2	.....	97	98	.....	255	Initialial	Accepté	Epsilons
0	0	0	0	0	0	0	0	0	**1**	0	**[1]**
1	0	0	0	0	0	0	0	0	0	0	**[2, 8]**
2	0	0	0	0	0	0	0	0	0	0	**[3, 6]**
3	0	0	0	0	**4**	0	0	0	0	0	-
4	0	0	0	0	0	0	0	0	0	0	**[5]**
5	0	0	0	0	0	0	0	0	0	0	**[8, 2]**
6	0	0	0	0	0	**7**	0	0	0	0	-
7	0	0	0	0	0	0	0	0	0	0	**[5]**
8	0	0	0	0	0	0	0	0	0	0	**[9]**
9	0	0	0	0	**10**	0	0	0	0	0	-
10	0	0	0	0	0	0	0	0	0	0	**[11]**
11	0	0	0	0	0	**12**	0	0	0	**1**	-

FIGURE 6 – Les tableaux de l'automate fini non déterministe pour l'arbre binaire **.(.(*(|(a,b)),a),b)**



	Transitions	Transitions avec "a"	Transitions avec "b"
1	{1,2,3,6,8,9}	{4,5,8,9,2,3,6,10,11}	{7,5,8,9,2,3,6}
2	{4,5,8,9,2,3,6,10,11}	{4,5,8,9,2,3,6,10,11}	{7,5,8,9,2,3,6,12}
3	{7,5,8,9,2,3,6}	{4,5,8,9,2,3,6,10,11}	{7,5,8,9,2,3,6}
4	{7,5,8,9,2,3,6,12}	{4,5,8,9,2,3,6,10,11}	{7,5,8,9,2,3,6}

FIGURE 7 – Les étapes de construire l'automate fini déterministe pour la RegEx **(a|b)*ab**

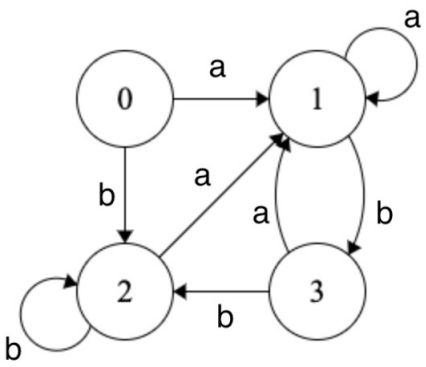


FIGURE 8 – L'automate Final pour la RegEx **(a|b)*ab**

4 Algorithmes de Jaccard

4.0.1 Distance de Jaccard

L'indice et la distance de Jaccard sont deux métriques utilisées en statistiques pour comparer **la similarité** et **la diversité** (en) entre des échantillons.

L'indice de Jaccard est le rapport entre le cardinal (la taille) de l'intersection des ensembles considérés

et le cardinal de l'union des ensembles. Il permet d'évaluer la similarité entre les ensembles. Soit deux ensembles A et B, l'indice est :

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|}$$

La distance de Jaccard, qui mesure la diversité entre les ensembles d'échantillons, est complémentaire du coefficient de Jaccard et est obtenue en soustrayant le coefficient de Jaccard de 1, ou, de manière équivalente, en divisant la différence des tailles de l'union et de l'intersection de deux ensembles par le taille de l'union :

$$d_J(A,B) = 1 - J(A,B) = \frac{|A \cup B| - |A \cap B|}{|A \cup B|}$$

4.0.2 Implémentation

L'algorithme du calcul de la distance de Jaccard se fait de la manière suivante :

1. Nous récupérons le cardinal de l'intersection entre les éléments présents dans un ensemble A et dans un ensemble B.

2. Ensuite nous récupérons le cardinal de l'union entre les éléments présents dans l'ensemble A et B.

3. Nous divisons la valeur récupérée de l'étape 1 par celle de l'étape 2 pour obtenir l'indice de Jaccard. On soustrait l'indice de Jaccard à 1 pour obtenir la distance de Jaccard pour 2 livres (A et B).

4.1 Matrice de Jaccard

A partir des distances, nous pouvons déterminer une matrice de Jaccard qui nous permettra de connaître l'ensemble des distances pour chaque livre de notre base de données.

4.1.1 Implémentation

Pour créer la matrice de Jaccard, nous réalisons ces 4 étapes :

1. Pour chaque livre (appelons les A), nous parcourons tous les livres (B) de la bibliothèque.

2. Pour chaque livre A et B, nous récupérons leur index associé (C'est-à-dire : On charge les fichiers d'indexage de chacun des livres. Le processus est triés rapide grâce à la sérialisation).

3. Nous calculons la distance de Jaccard pour chacun des livres. Pour gagner en temps de calcul, nous créons un cache. Si une distance a déjà été calculer précédemment, nous ne la recalculons pas et passons à la suite (exemple : Distance (1,2) == Distance (2,1)).

4. Une fois la matrice calculée, nous la sérialisons dans un fichier pour pouvoir la réutiliser plus tard.

4.2 Graphe de Jaccard

Un Graphe de Jaccard est un graphe comprenant des sommets et des arrêtes. Chaque sommet représente l'identifiant d'un livre de notre base de données. Chaque arrêtes correspond à la distance de Jaccard entre 2 sommets.

A la suite d'une réponse d'une recherche utilisateur, nous récupérons les voisins du sommet du graphe (=les livres voisins) les plus pertinents pour l'utilisateur. Les sommets représentent l'ensemble des livres (sous la forme d'un ID unique) et les arrêtes entre deux sommets sont créés si la distance de Jaccard n'excède par un certain coefficient. Cela signifie que certains sommets peuvent avoir aucun voisins. Ce sont des noeuds isolés.

4.2.1 Implémentation

La réalisation du graphe se présente de la façon suivante :

1. Nous créons tous les sommets en parcourant l'ensemble des livres de notre base de données. Chaque sommet associe l'identifiant d'un livre.

2. Pour chaque sommet, on calcule les distances avec les autres noeuds du graphe. Pour connaître la distance, on la récupère dans notre matrice de Jaccard crée précédemment.

3. Si une distance est inférieure au coefficient de Jaccard (ex : 0,7), nous ajoutons l'id du noeud à la liste des voisins de notre sommet.

Nous obtenons une complexité en $O(n^2)$ dans le pire cas.

5 Algorithme Closeness Centrality

Dans un graphe connexe, la Closeness Centrality d'un nud est une mesure de la centralité dans un réseau, calculée comme l'inverse de la somme de la longueur des chemins les plus courts entre le nud et

tous les autres nuds du graphe. Ainsi, plus un nud est central, plus il est proche de tous les autres nuds.

$$C(x) = \frac{1}{\sum_y d(y,x)}$$

où d(y,x) est la distance entre les sommets x et y.

5.1 Implémentation

Notre algorithme Closeness Centrality permet d'obtenir l'ensemble des rangs associés aux indices des livres suite à une recherche utilisateur. C'est à dire un classement des livres les plus pertinent pour l'utilisateur.

Notre algorithme se décrit de la façon suivante :

1. Nous parcourons l'ensemble des identifiants des livres retournés par une recherche d'un utilisateur.
2. Pour chaque identifiant, nous parcourons les sommets du graphe de Jaccard en additionnant les distances entre cette identifiant et les sommets de notre graphe. Nous récupérons les distances grâce à la matrice de Jaccard.
3. Ensuite, nous associons à chaque sommet, un rang qui est égal à l'inverse des sommes des distances calculées à la deuxième étape.

6 Test de performances

Pour tester les performances des algorithmes expliqués précédemment, nous utiliserons plus que 1600 livres téléchargés à partir de site Gutemberg. Chaque livre contient plus que 10000 mots.

6.1 Les fonctionnalités de recherche

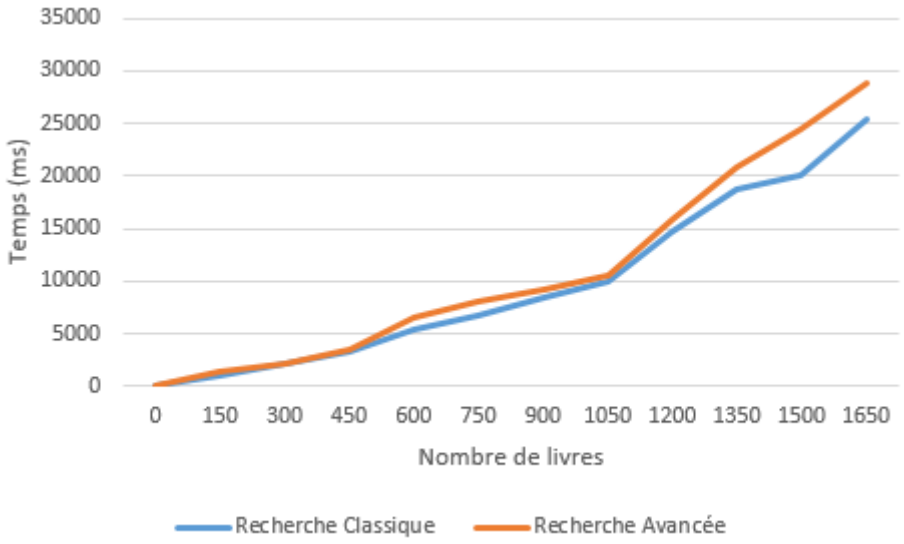


FIGURE 9 – Diagramme de performance pour la recherche classique et avancée

On remarque une croissance constante. Le temps de recherche est presque proportionnel aux nombres de livres présent dans la base de données.

6.2 Les algorithmes de Jaccard

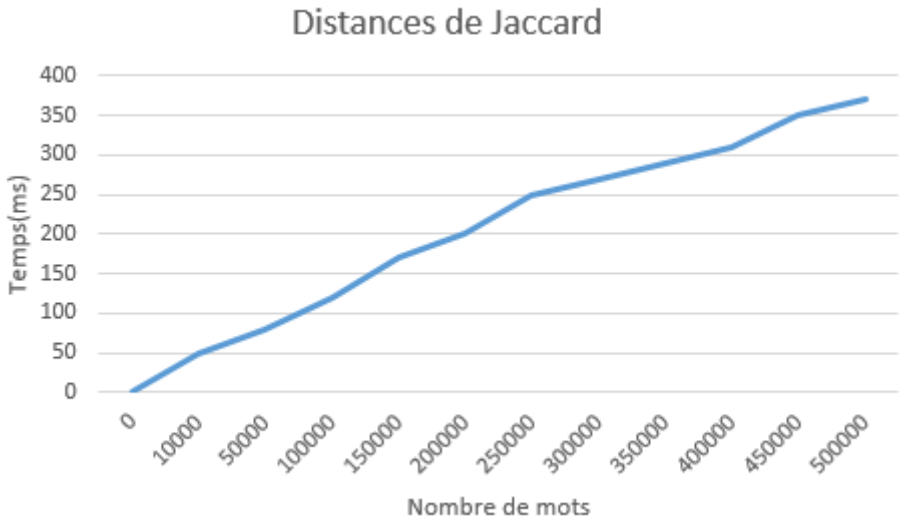


FIGURE 10 – Digramme de performance pour la distance de Jaccard

On remarque dans un premier temps que le temps de calcul d'une distance est pratiquement linéaire.

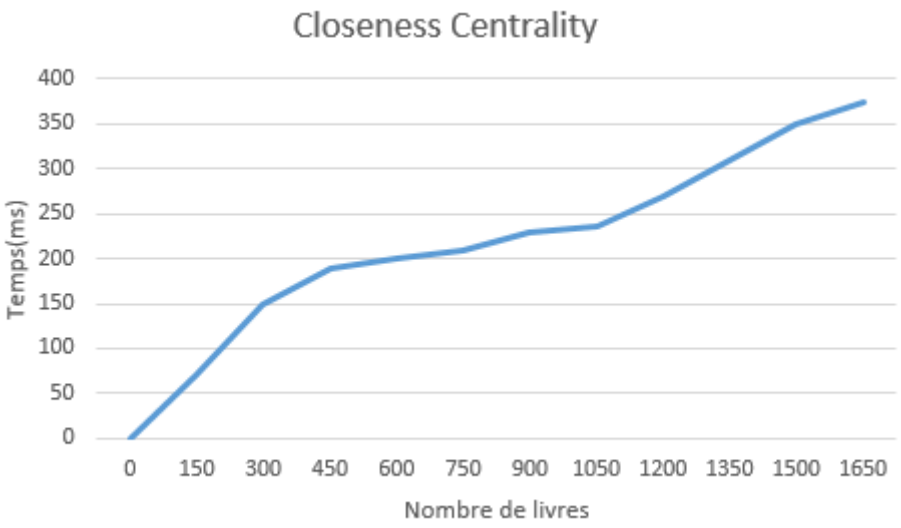


FIGURE 11 – Diagramme de performance pour closeness Centrality

6.3 L'algorithme Closeness Centrality

On remarque une augmentation rapide entre 150 et 450 livres. En effet, il est possible que les livres compris dans cet intervalle soit tous dans le graphe.

7 Conclusion

Notre application web permet aux internautes de chercher des livres à l'aide d'un mot clé ou bien une expression régulière. Ainsi que notre application trie les livres par plusieurs critères nombre d'occurrence de la mot recherché et elle propose des suggestions de livre qui peuvent être intéressantes pour l'internaute.