Rapport - MAAIN2020

Étienne Jouanne, 21500537 - Rhodier Pierre, 21500981 01/05/2020

Introduction

Dans ce rapport, nous parlerons de nos choix techniques qui ont pu être choisis pour nos calculs. Nous détaillerons également le fonctionnement des fonctions.
Nous ne parlerons pas ici de la partie serveur.

Architecture

Tout d'abord, nous avons choisi de faire la partie calcul dans le langage Java. Nous avons hésité pendant un moment à faire les calculs en Python, car nous pensions que ce langage était le plus simple et le plus rapide en temps d'exécution des programmes pour un nombre assez large de données (avec notamment les librairies de machine learning disponible par exemple). Mais après plusieurs recherches sur internet, il nous est apparu que le fait que Java soit asynchrone contrairement à Python augmente légèrement la vitesse de calculs.
De plus, Java est un langage sur lequel nous sommes très familiers contrairement aux autres langages qui nécessitent d'apprendre des méthodes d'encodage ou même d'apprendre un langage. Nous n'avions donc pas le temps de nous perfectionner sur qui sont bien plus rapides que Java.

Nous avons pu créer différentes class pour produire notre moteur de recherche :

- CLIList : la version liste chaînée du format CLI.
- CLIArray : la version tableau du format CLI.
- Vecteur : l'implémentassions des vecteurs.
- Page : qui nous aide à trier les pages en fonction de leur note de Page Rank.

Au début, nous avions pensé à créer une interface CLI qui aurait été implémentée par CLIList et CLIArray. Mais après quelque temps, nous nous sommes aperçu qu'il y avait trop de différences entre les classes : il n'y avait presque aucune fonction qui était implémentée dans les deux classes. Nous avons donc décidé de supprimer l'interface CLI qui embêtait plus qu'autre chose et deux faire deux class distincts. Néanmoins, les tableaux prennent beaucoup d'espaces de stockage que les listes chaînées. Il est donc possible de convertir un objet CLIList en CLIArray.
Nous avons également créé notre propre class de liste chaînée (MyList et Noeud). Nous avons également remarqué que faire notre propre liste chaînée prenait beaucoup moins d'espace de stockage qu'une liste chaînée déjà implémenter comme LinkedList ou ArrayList.

Fonctions utilisés

Les fonctions décrites en dessous sont des fonctions utilisées pour calculer le Page Rank des pages. Ces fonctions se trouvent dans la class Projet.

1. Select Title

Cette fonction est utilisée pour avoir un premier ensemble de mots. Nous avons décidé de sélectionner les mots des titres et qui ne font pas partie des mots interdits. Nous parcourons tout d'abord le fichier de mots interdits (fw). Nous mettons tous les mots dans un HashSet. Ensuite, nous parcourons notre premier ensemble de pages. Quand nous trouvons une ligne du texte qui contient les balises alors nous supprimons les balises, prenons les mots du texte et nous les ajoutons à un fichier .txt. Nous faisons bien attention de ne pas prendre deux fois un même mot pour éviter de trop perdre d'espace de stockage inutilement.
Au passage, nous enregistrons les titres en entier dans un autre fichier. Ces données nous sont utiles pour plus tard pour gagner en efficacité.

2. Most Counted Words

Dans cette fonction, nous créons notre vrai ensemble de mots. Nous avons décidé de choisir de prendre les 10 000 mots les plus fréquents dans les textes qui sont présents dans l'ensemble des mots du titre que nous avons pu faire à la fonction précédente. Ce que nous faisons est d'abord de trouver tous les textes, ensuite de sélectionner chaque mot, de vérifier chaque mot s'il était dans la liste de mots des titres et de l'ajouter à une nouvelle liste s'il y ait inconnu ou alors d'incrémenter sa fréquence si ce mot a déjà été rencontré. À la fin, il nous suffit de trier cette liste et de prendre les 10 000 mots les plus fréquents.

3. Words Pages and Links

Avec la fonction précédente, nous avons pu trouver les 10 000 mots les plus fréquents dans les pages. Dans cette fonction Words Pages and Links, nous allons compter la fréquence des mots les plus fréquents sur chaque page. Au passage, nous pourrons également détecter les liens qui sont dans les textes afin de les répertorier.
Tout d'abord, nous récupérons la liste des mots les plus fréquents que nous avons pu avoir à la fonction précédente. Nous récupérons également la liste des titres que nous avons pu faire lors de la première fonction. Ensuite, nous parcourons les textes de chaque page. Nous prenons chaque mot des textes et nous augmentons sa fréquence de 1. Au passage, si ce mot est entre double crochet ("[[...]]"), cela signifie que c'est un lien. Nous vérifions alors que ce lien est bien dans la liste des titres que nous avons, et si c'est bien le cas, alors nous enregistrons la connexion dans une class CLI (CLIList car nous ne connaissons pas a l'avance la longueur que peut avoir la taille du CLI).
Enfin, nous obtenons alors deux objets :

- une matrice sous forme de HahsMap avec en ligne les mots les plus fréquents et en colonne

les pages où ces mots ont pu être détectés. Ce HashMap va être ensuite converti sous forme CLI.

- un tableau sous forme CLI qui a pour ligne les pages, en colonne les pages et comme valeur 1 si la page de la ligne contient un lien de la page de la colonne.

Nous enregistrons ces deux CLI dans un fichier pour avoir une sauvegarde.

4. Tri

Enfin, après avoir eu la relation mot-page et la relation des liens, les deux sous forment CLI, il nous reste à calculer le Page Rank de chaque page et de trier les pages en fonction de cette note. Nous récupérons d'abord les deux CLI (en CLIArray cette fois pour avoir le moins d'espace), nous calculons ensuite le vecteur du Page-Rank zéro. Nous avons pu utiliser la variante du Page-Rank avec le zap. Une fois le calcul terminé, nous obtenons une note pour chaque page. Finalement, nous regardons d'abord les relations mot-pages, pour chaque mot nous regardons la fréquence du mot pour chaque page et si celle-ci est assez élevée, nous pouvons ajouter cette page à une liste chaînée de class Page. Nous ajoutons également la note du Page-Rank obtenu pour la page. Nous trions la liste chaînée en fonction de la note du Page Rank de manière décroissante.
À la fin, nous obtenons une relation mot-page sous forme CLI avec en ligne, les mots et en colonne, les pages et comme valeur la fréquence des mots sur chaque page. De plus, chaque ligne est triée en fonction du Page-Rank des pages. Nous obtenons alors pour chaque ligne en premier les pages aillant la meilleure note des Page-Rank avec une fréquence minimale.

Conclusion

Pour conclure, sur la machine lulu de l'UFR, en utilisant la commande time (en real), nous avons ces résultats avec les corpus suivants :

- **football.xml** : 151 263 pages, les calculs mettent environ 5m53,826s.
- **cinema.xml** : 196 402 pages, les calculs mettent environ 9m11,673s.
- **7emeart.xml** : 276 571 pages, les calculs mettent environ 12m8,648s.

Il est difficile de pouvoir comparer, mais nous trouverons que ces résultats sont très corrects. De plus, quand nous effectuons des requêtes sur notre serveur, le moteur de recherche fonctionne plutôt bien. La vitesse du serveur est aussi très satisfaisante : la plupart des requêtes se font en moins d'une milliseconde. Nous sommes assez fières de notre résultat.

Comme vous l'avez peut-être remarqué, nous avons essayé de ne parcourir qu'une seule fois le corpus à chaque fois que nous en avions besoin (notamment pour la relation mot-page et la recherche du lien qui se fait en une seule fois). Nous avons pensé que faire cette recherche était le plus long à faire. Alors un des moyens les plus rapides était pour nous de sauvegarder certaines informations dans des fichiers pour les retrouver plus facilement (comme les titres) et ensuite de reparcourir ces fichiers et enregistrer ces informations dans des HashMap ou HashSet suivant

Figure 1: Recherche Star Wars

Figure 2: Recherche Titanic

Figure 3: Recherche Charlie Chaplin

lequel nous avions besoin.

Les tableaux sont aussi une solution pour augmenter la rapidité des programmes. Mais ceux-ci sont assez compliqués à utiliser en Java. C'est pour cela que dans certains cas, nous ne connaissons pas à l'avance la taille d'un tableau. C'est pour cela que nous utilisons des listes chaînées au départ et ensuite nous convertissons vers des tableaux.