Rapport : projet MAAIN

Projet réalisé par: Oussama AOUESSAR, Ilyesse Mohamed MEDJADI et Lilya MELILA

Description

Ce projet a pour but de parser Wikipédia (version française) pour en extraire un corpus contenant un certain nombre de pages autour d'un sujet. Ici nous avons décidé de construire notre corpus sur le thème de l'informatique.

Une fois le corpus récupéré, le projet repose dessus. Nous avons donc codé un moteur de recherche qui, à partir de notre corpus et d'une recherche lancé par un utilisateur, nous permet de récupérer la liste des résultats et de l'afficher, pour qu'en cliquant sur l'un des résultats, l'utilisateur soit redirigé automatiquement vers la page Wikipédia correspondant. Nos résultats sont triés grâce au score que le pagerank leur attribue. Notre pagerank utilise un facteur zap $d = 0.1$ et une précision $\epsilon = 10$ car après plusieurs tests, ces valeurs sont celles qui nous semblent les plus appropriés.

Choix et Motivation

Nous avons fait le choix de coder en Node JS pour pouvoir inclure nos pages web tout au long de notre projet. Nous voulions retravailler nos acquits mais aussi apprendre de nouvelles choses, c'est pour cela que le choix du Node JS nous a paru judicieux.

En ce qui concerne la génération du corpus, nous nous sommes mis d'accord qu'en tant qu'informaticiens, il était inenvisageable que notre corpus ne contienne pas quelques notions d'informatique. On s'est donc tourné par un sujet très en vogue ces temps-ci, l'intelligence artificielle. C'est pourquoi notre corpus regorge d'algorithmes en tous genres.

Pour la création du graphe et du collecteur, nous nous sommes rendu compte que le parcours systématique du corpus entier n'était pas envisageable... personne n'attendrait plus d'une heure pour avoir un résultat de recherche. Nous avons donc décidé de faire un parcours unique de corpus.xml qui puisse nous permettre de récupérer le graphe pour le stocker dans le fichier graph.xml et récupérer le collecteur pour le stocker dans le fichier collecter.xml. Les créations respectives de ces deux fichiers se feront à l'aide des fichiers graph generator.js et dictionarygenerator.js.

Du point de vue des recherches, parcourir tout le corpus.xml est beaucoup trop long, comme dit juste avant. Si, pour chaque recherche, nous devions parcourir le fichier page par page, il nous aurait fallu attendre plus de 30 minutes afin de pouvoir récupérer un résultat. Ceci étant "un peu long", nous avons donc choisi une autre solution... le collecteur ! Il contient les 10.000 mots les plus fréquents dans le corpus, on base donc notre recherche sur la relation mot-page calculé dans notre serveur. Cela nous permet d'avoir un résultat de recherche plus rapide qu'en parcourant tout le corpus.xml, même si elle est plus limitée.

Répartition et organisation du travail

En ces temps difficiles, nous avons dû trouver une solution efficace pour pouvoir, malgré la distance, échanger et coder ensemble.

La majorité de nos échanges ce sont donc fait par téléphone, et des sessions de codage étaient organisées via Discord. Discord ayant une fonctionnalité de partage d'écran, nous pouvions coder ensemble et nous entraider lorsque l'un de nous rencontrait des difficultés.

Au niveau de la répartition du travail, comme nous vous l'avons déjà dit, la majorité du temps nous codions ensemble via le logiciel Discord. La répartition se faisait en fonction des besoins du moment, si une personne était en difficulté ou si elle avait besoin de plus de précision sur une partie du code une autre personne venait l'aider afin de travailler en utilisant la méthode agile "pair programming" où l'un va ecrire et l'autre va l'aider dans la reflexion et résolution des problèmes.

Pour ce qui est de l'organisation, nous nous sommes d'abord concentrées sur la génération du corpus, puis sur la création du graphe et de la matrice CLI. En parallèle, le calcul du collector et pour finir le Page Rank.

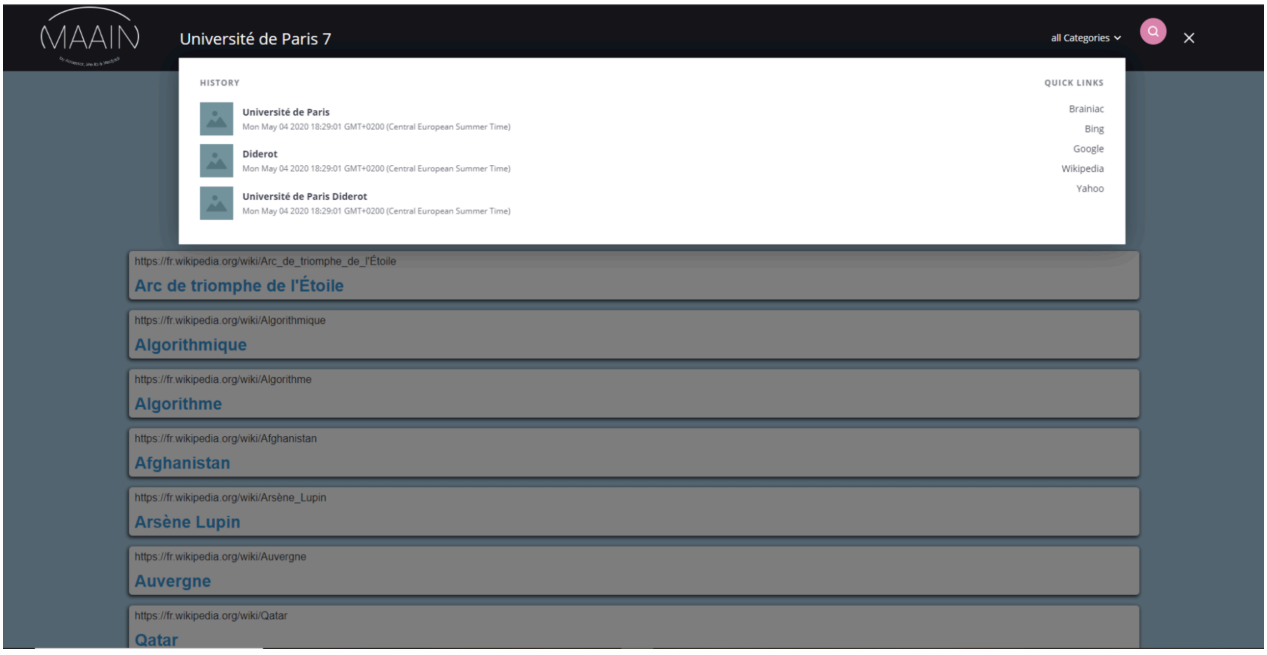
Pour le front, la gestion de index.ejs et du CSS c'est fait au fur et à mesure.

Démonstration

Afin d'effectuer une recherche sur notre site, il faut cliquer sur la loupe sur le bord en haut a droite de la page, une barre de recherche s'ouvrira et vous suggérera vos premières recherches.

Lors d'une recherche, nous vous affichons une liste de pages contenant 10 resultats par page. Les liens sont cliquable et peuvent vous mener vers la page Wikipedia en rapport avec le titre du lien.

Après avoir effectué votre première recherche, nous commencer a stocker vos recherches et nous vous les affichons ensuite dans un historique.



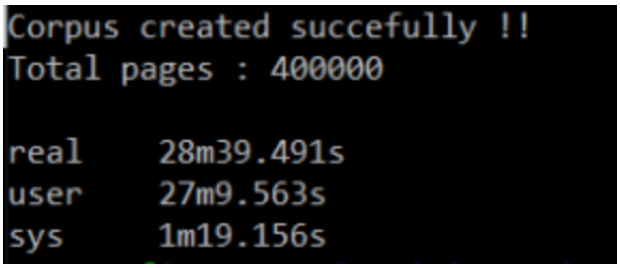
Analyse du temps

Les temps que nous pouvons avoir ici, dépendent de la puissance de calcul de la machine. Nous avons utilisé un ordinateur assez vieux afin de vous montrer des valeurs temporelles réelles.

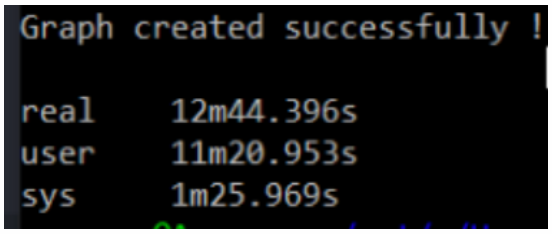
- Dictionarygenerator.js: Le temps total pour construire notre dictionnaire et en parallèle faire la relation mot-pages afin de générer le Collector pour tout le corpus (un total d'environ 600.000 mots appris) est d'environ une heure:

Ici, chaque mot ne peut avoir plus de 1500 pages liées à lui car pour des raisons d'optimisation mémoire nous avons dû opter pour cette limite afin que le projet soit utilisable par tous les ordinateurs. Il nécessite un maximum de 6Go de RAM disponible.

- Corpusgenerator.js: Il parcours le fichier frwiki (~20Go) afin de récupérer les 400.000 premières pages en relation avec les domaines de l'informatique, de l'Intelligence Artificielle et de la robotique. À sa création, le corpus pèse ~2.5Go et prend ici 28 minutes.



- Graphgenerator.js: Il parcours le corpus que nous avons initialement généré, afin de construire notre graphe. Les nœuds correspondent aux pages et leur liste d'adjacences, elle, correspond aux liens de pages liés à cette dernière (s'il y a un arc les liants).



Difficultés rencontrées

L'une des principales difficultés rencontrées a été de pouvoir récupérer les donner en essayant de prendre le moins de temps possible. Le parcours du fichier corpus.xml est très long et demande beaucoup de ressources à chaque parcours. Nous avons pu gérer ça en faisant le choix de le parcourir uniquement deux fois, la première fois pour la création de graph.xml et la deuxième fois pour générer collector.xml.

Un autre point compliqué a été de pouvoir coder à distance. On a fait le choix de coder au maximum ensemble ce qui implique de se fixer des horaires de travail où l'on sera tous disponibles, ce qui n'est pas toujours facile, réussir à être à jour sur le travail de chacun, se répartir des tâches. Toutes ces contraintes, nous avons pu dans la globalité les gérer.

Notre programme demande une certaine capacité de mémoire vive, ce qui nous a dû quelques belles frayeurs étant donné que nous n'avons pas tous des ordinateurs très performants. Nous avons dû passer un certain temps a optimiser nos algorithmes afin que le projet soit viable en temps, tout en économisant de la mémoire afin qu'il soit utilisable par n'importe quelles machines.

Un membre de notre groupe (Ilyesse) a été atteint du Covid-19, ce qui lui a voulu d'être indisponible un certain temps. Nous avons donc palier à son absence, jusqu'à son rétablissement où il a pu revenir travailler avec nous. Il est, aujourd'hui, en bonne santé

Connaissances acquises

- Nous avons eu l'occasion d'approfondir nos connaissances en NodeJS, technologie auquel, nous avons déjà eu une initiation en L3.
- Plusieurs nouveaux algorithmes ont été abordés dans notre projet:
  - La distance de Levenstein afin de calculer la racine des mots
  - les matrices CLI afin régler le problème des matrices creuses
  - les differents algorithmes du Pagerank
    - pagerank_eps(...): utilisant une précision epsilon
    - pagerank_zap_steps(...): utilisant un facteur zap avec un nombre de pas
    - pagerank_zap_eps(...): utilisant un facteur zap et une précision epsilon
  - les algorithmes de stream, nous permettant de lire de lourds fichiers sans pour autant surcharger la mémoire de la machine.
- Comme dans le module *Grand Réseau d'interactions* de M. Montgolfier, nous nous sommes familiarisés avec la gestion de grosses quantités de données tout en gardant un temps d'exécution correcte.
- Les services WEB, plus particulièrement les services GET et POST que nous avons utilisés afin de transmettre les données entre le serveur et le navigateur.