Rapport de soutenance 2

Quentin Couavoux
Léo Menaldo
Alexandre Colliard
Justine Gueulet

Table des matières

1 Plan de soutenance

Introduction

Démonstration

- User Input Processing
- Crawler
- Scraper
- Interface

Conclusion

- Éléments terminés
- Éléments commencés
- Éléments à commencer

2 Prélude

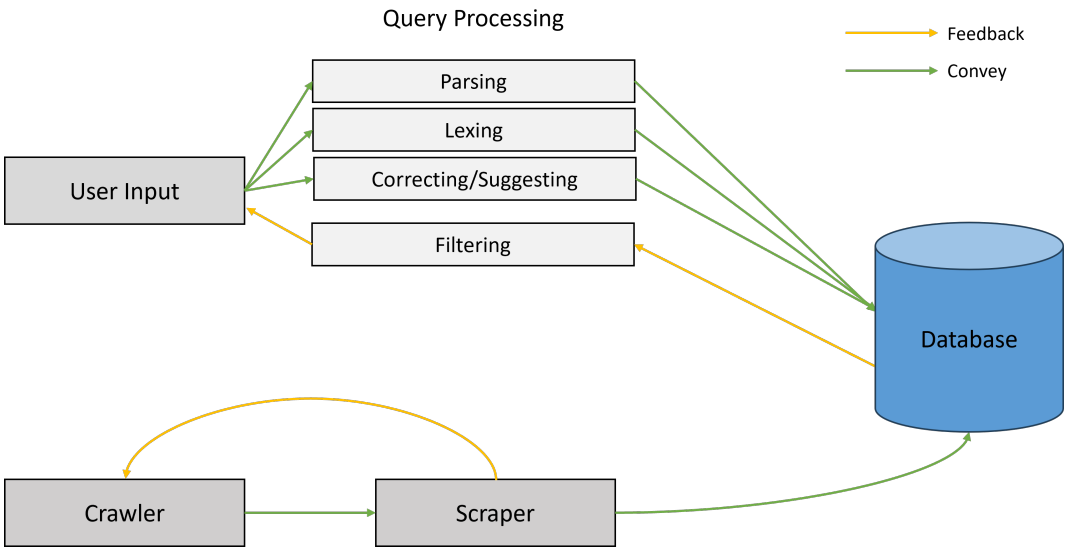
2.1 Le projet

En 1990, Alan Emtage crée le premier moteur de recherche afin de faciliter la navigation sur internet pour les utilisateurs. Depuis, plusieurs moteurs de recherche étant fondés sur le travail d'Alan Emtage ont été conçus pour aider les utilisateurs à faire leurs recherches aisément tout en répertoriant les sites web sur Internet. Or, malgré les avancements technologiques de ces moteurs de recherche, seulement 85% des utilisateurs obtiennent des informations précises qui répondent à leur demande. Notre groupe a donc décidé de s'intéresser à ce problème.

Pour notre projet, nous introduisons un moteur de recherche stable et rapide, tout en répondant de manière précise aux requêtes des utilisateurs.
Un moteur de recherche peut être séparé en deux parties principales : celle du "User Input" et celle du "Crawler". Ces parties fonctionnent simultanément afin de permettre à l'utilisateur d'effectuer ses recherches tout en agrandissant la base de données.

Le moteur de recherche permet à l'utilisateur de rechercher des informations via des requêtes. Ces requêtes sont ensuite gérées par le "Query Processing" où les mots clés de la requête sont déterminés. Il est possible de proposer des suggestions et/ou des corrections d'orthographes en cas de mots clés mal écrits ou erronés afin de permettre à l'utilisateur d'obtenir de meilleurs résultats. Une fois la requête traitée, les sites Web répondant au mieux à la demande de l'utilisateur sont récupérés de la base de données, triés en fonction de leur pertinence et envoyé à l'utilisateur.

La partie "Crawler" possède deux catégories : le "Crawler" et le "Scraper". Le Crawler est le processus de recherche et de récupération itérative de liens de sites Web à partir d'une liste d'URL de départ. Le Scraper, quant à lui, récupère les liens visités pour extraire et d'envoyer au Crawler les URLs référencées dans ces derniers pour continuer l'exploration Web. Le Scraper se charge également de traiter les documents Web et d'en extraire des informations afin d'y envoyer à la base de données.



3 Présentations

3.1 Quentin Couavoux

Bonjour, je me présente en tant qu'étudiant en informatique, actuellement en deuxième année à l'EPITA. Mon intérêt pour l'informatique remonte à l'âge de 10 ans, une époque où je m'initiais déjà à la programmation sur Minecraft. Cette passion m'a accompagnée tout au long de mon parcours, m'incitant à m'engager davantage dans ce domaine. À l'heure actuelle, je suis impatient à l'idée de débuter le projet de S4, suite logique des projets de S2 et S3 qui se sont déroulés de manière singulière et enrichissante. Chaque étape m'a permis de développer mes compétences et d'explorer divers aspects de l'informatique. Ce qui me motive particulièrement dans le projet actuel, c'est l'aspect algorithmique. La perspective de travailler sur des algorithmes complexes. Ainsi, je suis prêt à relever les défis et à contribuer pleinement au succès de ce projet. Que l'aventure commence.

3.2 Léo Menaldo

Joueur de jeux vidéo depuis mon plus jeune âge, j'ai très vite su que j'allais m'orienter vers l'informatique plus tard. Les années ont passé et j'ai intégré l'EPITA. Cela fait maintenant une année et demie que j'y suis et que je m'y investis, les moments les plus mémorables restent les périodes de projets. Que ce soit le jeu vidéo fait durant le deuxième semestre ou le terrible solveur de sudoku du troisième, j'ai su surmonter ces épreuves en restant soudé avec mon groupe. Ce projet s'annonce comme quelque chose de coriace qui va nous défier sûrement bien plus que les précédents projets.

3.3 Alexandre Colliard

Passionné d'informatique depuis la seconde, je me suis entraîné dans de nombreux projets ambitieux, seul ou avec des amis. J'ai déjà codé un jeu Android multijoueur, que j'ai pu déployer sur le Play Store, mais il n'a jamais vraiment abouti. Le projet S2 a été pour moi un moyen de découvrir la gestion de projet dans un cadre sérieux et professionnel (ici scolaire). Le projet S4 sera l'occasion de pousser l'aspect algorithmique des applications et d'appliquer les notions d'optimisation et de rapidité. J'espère que notre expérience sur les différents projets de groupe (S2,S3) nous permettra de mieux gérer notre temps pour finir l'application. Marge !

3.4 Justine Gueulet

Depuis mon plus jeune âge, le monde de l'informatique m'a toujours intéressé. Mon père travaillant dans l'informatique, je me suis souvent amusée à l'imiter. Cette passion n'a fait que croître lors de la spécialité NSI, surtout l'intelligence artificielle et la robotique. J'ai toujours voulu faire un projet qui se concentre sur les moteurs de recherche avec du Natural Language Porcessing (NLP). Or, il faut tout d'abord avoir des connaissances sur le fonctionnement des moteurs de recherche. C'est pour cela que j'ai vraiment hâte de commencer ce projet avec mes camarades de classe !

4 Assignation des tâches

Ci-dessous, les tâches assignées à chacun des membres du groupe :

4.1 Quentin Couavoux

Pour cette soutenance, cette partie doit permettre de faire des requêtes SQL dans la base de donnée.

4.2 Léo Menaldo

Pour cette soutenance, il doit être possible de parcourir le web en partant d'une seule adresse.

4.3 Alexandre Colliard

Pour cette soutenance, cette partie doit pouvoir suggérer des propositions de fin de recherche à l'utilisateur. L'interface doit aussi être commencée pour avoir un rendu visuel.

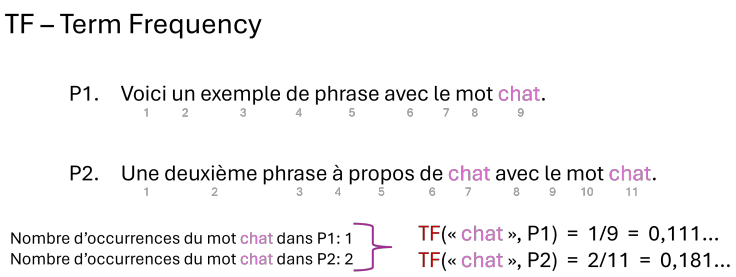
4.4 Justine Gueulet

Pour cette soutenance, la partie concernant le Scraper devait être capable d'utiliser l'algorithme du TF-IDF pour extraire les mots clés d'une page Web.

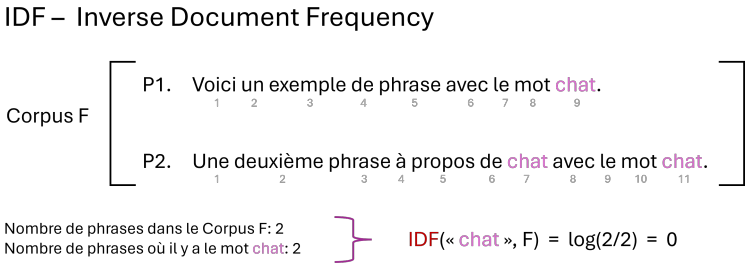
5 Nettoyage des données - Scraper

5.1 TF-IDF

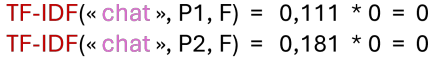
Le "TF-IDF", aussi connu sous le nom de "Term Frequency * Inverse Document Frequency", est une méthode de SEO (Search Engine Optimization) permettant de trouver les mots les moins utilisés sur une page Web aisni que les mots les plus utilisés. Pour ce faire, il est séparé en deux parties distinctes : le TF (Term Frequency) et le IDF (Inverse Document Frequency).



Le TF permet de calculer la fréquence d'un mot dans un document, donc dans notre cas dans une page Web. Il va compter le nombre d'occurences du mot pour ensuite le diviser par le nombre de mots en tout sur la page Web. Faire cela permet de voir si un mot est important au sein d'une certaine phrase ou non, ainsi que de voir sa pertinence globale sur la page Web. Or cela ne suffit pas pour savoir si ce mot en particulier est un mot clé. C'est pour cela que nous devons utiliser la partie IDF.



La partie de l'IDF s'occupe de regarder la rareté du mot choisi dans un corpus, soit dans notre cas une page Web. Il commence par décompter le nombre de phrases dans ce corpus, ainsi que le nombre de phrases dans lesquelles le mot est trouvé. Le nombre de phrases contenant le mot est ensuite divisé par le nombre de phrases au total, puis la formule mathématique du logarithme est appliqué au résultat obtenu. Le TF-IDF multiplie donc la partie du TF à celle de l'IDF, ce qui permet de voir le poids de ce mot dans la page Web. Pour revenir sur notre exemple, voici les résultats obtenus pour le mot "chat" :



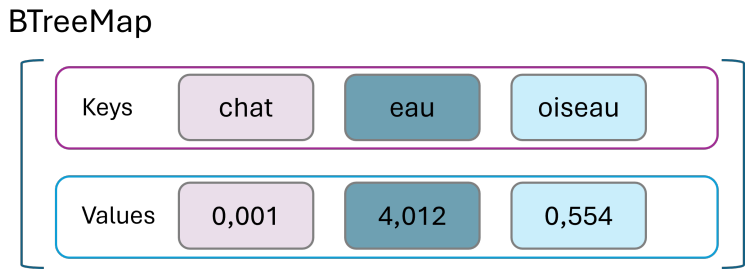
Plus un résultat TF-IDF est élevé, plus cela signifie que le mot est rare et donc potentiellement un mot clé. L'algorithme du TF-IDF est très pratique pour notre projet or sachant qu'il faut parcourir plusieurs fois une même page Web pour obtenir toutes ces variables, le programme peut être assez lent.

5.2 Optimisation TF-IDF — BTreeMap

Comme mentionné précédemment, l'algorithme du TF-IDF peut être assez lent, surtout si l'on décide de garder en mémoire tous les résultats obtenus. La méthode la plus naïve pour appliquer le TF-IDF consiste à parcourir plusieurs fois le document, en gardant dans des Vecteurs ou des Arrays les nombres d'occurrences de chaque mot, les différents phrases du texte etc. Même en arrivant à tout faire en un seul parcours, cela reste peu optimisé et inexploitable. Pour éviter cela, il est donc nécessaire d'utiliser des structures de données optimisées permettant de stocker les résultats tout en étant plus rapide qu'un simple vecteur. En Rust, il existe trois grandes structures de données aidant à réaliser ceci :

— les HashMaps,
— les HashSets,
— et les BTreeMaps.

Les HashSets, bien que très pratique pour leur rapidité de recherche et leur efficacité, ne sont pas utiles dans le cas du TF-IDF car ils ne peuvent stocker qu'une seule valeur et l'accès aux éléments est imprévisible avec des modèles d'itération (boucles for, etc).
Les HashMaps sont des dictionnaires facilement utilisables qui possèdent également une rapidité de recherche remarquable. Or, ces structures ne sont pas ordonnées et peuvent performer médiocrement dans certaines situations allant jusqu'à une complexité de $O(n)$.
La dernière option possible est la structure de données BTreeMaps. Cette dernière permet non seulement de conserver plusieurs valeurs, mais reste ordonnée. Sa complexité est de $O(log(n))$, ce qui représente plus que la complexité des HashMaps dans les meilleurs cas ($O(1)$), or les BTreeMaps restent stables et ne dépassent pas $O(log(n))$ même dans les pires situations, ce qui est très pratique pour le TF-IDF qui parcourt des pages Web pouvant posséder jusqu'à plusieurs dizaines de milliers de mots.



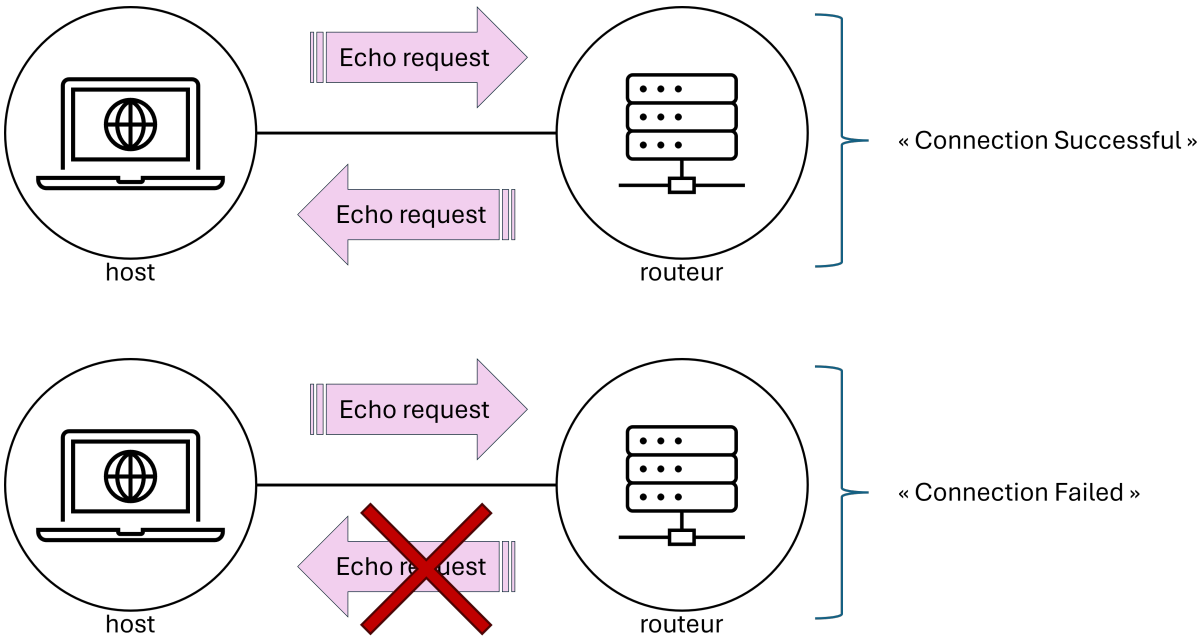
Nous utilisons donc des BTreeMaps afin d'optimiser le TF-IDF : ces derniers nous permettent de garder en mémoire le mot clé en tant que "Key" ainsi que son résultat TF-IDF en tant que "Value". Les BtreeMaps sont donc utilisés comme des dictionnaires ordonnés et optimisés. Malgré les optimisations, le TF-IDF reste énorme au niveau de la quantité de données à traiter, ce qui peut (en fonction de la taille de la page Web) rendre la fonction un peu longue. Malheureusement ceci reste un des problèmes de l'algorithme du TF-IDF que nous ne pourrons pas résoudre. En effet le TF-IDF reste un algorithme possédant une approche primitive qui ne prend pas en compte certains facteurs tels que les synonymes, les intentions de recherches ainsi que les objectifs de rédaction.

5.3 Récupération HTML à partir de l'URL

Le Scraper est une partie qui est censée récupérer le code source en HTML d'une URL pour pouvoir ensuite l'analyser et appliquer certains algorithmes sur ce dernier. Or, lors de la première soutenance, la partie du Scraper a rencontré quelques problèmes de librairies Rust et n'arrivait pas à implémenter correctement la fonction permettant la conversion. La librairie que nous avions essayé d'utiliser était la librairie HTTP Hyper qui est rapide et stable. Or c'est aussi une librairie très low-level qui est assez compliquée à prendre en main d'où notre incapacité à faire marcher notre fonction de conversion.
Après plus de recherche, sachant que cela n'était pas une priorité pour notre première soutenance, nous avons trouvé plusieurs librairies HTTP Rust permettant d'effectuer ce que nous voulions faire de manière toute aussi rapide et beaucoup plus simple. La librairie HTTP que nous avons utilisé pour le Scraper est donc la même librairie que nous utilisons pour le Crawler, ce qui nous évitera d'avoir plusieurs librairies servant la même nécessité.

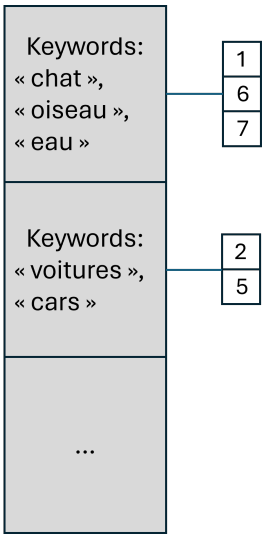
Une fois la fonction implémentée, lorsque nous n'avons pas de connexion internet la fonction crée une boucle infinie ce qui pose problème. Il a donc fallu implémenter une fonction supplémentaire afin de vérifier si nous captons une connexion internet. Si nous sommes connectés à un réseau, la fonction de conversion d'une URL au code HTML se lance sans soucis, sinon nous obtenons une erreur immédiatement nous prévenant de notre manque de connexion internet. Pour cela, nous faisons appel au protocole ICMP (Internet Control Message Protocol), où l'hôte et les routeurs échangent des données avec l'hôte qui renvoie une erreur si les données n'ont pas été reçues intactes.

Cette fonction rajoute une sécurité afin d'éviter que notre moteur de recherche émette des erreurs à cause d'un manque de connexion qui crée une boucle infinie.



5.4 Organisation de la base de données

L'implémentation de l'algorithme du TF-IDF s'est terminée en avance ce qui nous a permis de prendre un peu d'avance sur la $3^{ème}$ soutenance en ce qui concerne notre base de données. La base de données est au coeur de ce projet, car sans cette dernière, les parties du Crawler & Scraper et du User Input Processing ne pourraient pas communiquer. Pour un moteur de recherche, une base de donnée ayant pour structure celle du "Invertex Index" est nécéssaire afin d'optimiser et de rendre plus efficace les recherches et requêtes SQL à effectuer sur cette dernière.



Nous aurons une première base de données avec les mots clés ainsi que des indexes qui possèdent une correspondance dans notre deuxième base de données.

1	https://les-chats.com	fr	« Les chats »	2018
2	https://voitures.com	fr	« L'histoire des voitures »	2020
3	https://online-shopping.com	en	« shopify »	2011
4	https://les-chapeaux.com	fr	« les chapeaux »	2003
5	https://cars.com	en	« cars »	2003
6	https://chats-&-oiseaux.com	fr	« Oiseaux et chats »	2023
7	https://les-oiseaux-&-l'eau.com	fr	« Les oiseaux boivent ? »	1998
...	...	...	...	...

Dans notre deuxième base de données, les indexes de notre première base de données donnent des informations sur le site, telles que sa data, son titre, sa langue ainsi que son URL. Nous implémenterons ces deux bases de données dans des fichiers SQLite afin que la partie du User Input Processing puissent accéder aux bases de données sans soucis via des requêtes SQL.

6 Crawler

6.1 Qu'est ce qu'un crawler

Pour que l'on puisse effectuer des recherches sur le web, il est important d'avoir des sites web à indexer ! Pour cela, il faut donc récupérer leurs adresses, ainsi que leur contenu.
C'est ainsi que l'on trouve une utilité au "crawler", l'objectif est simple, en partant d'un site, on récupère tous les sites qui y sont référencés pour les visiter puis on répète ce processus autant de fois que l'on veut.

6.2 Fonctionnement

6.2.1 Créer une requête HTTP

Comme vu précédemment dans des travaux pratiques effectués durant le semestre dernier, on sait qu'accéder à un site web ne se fait pas en un clic...
En effet, on ne peut pas simplement envoyer l'adresse d'un site web au serveur et espérer qu'il nous renvoie le contenu de celui-ci.
Voici à quoi une requête HTTP est censée ressembler :

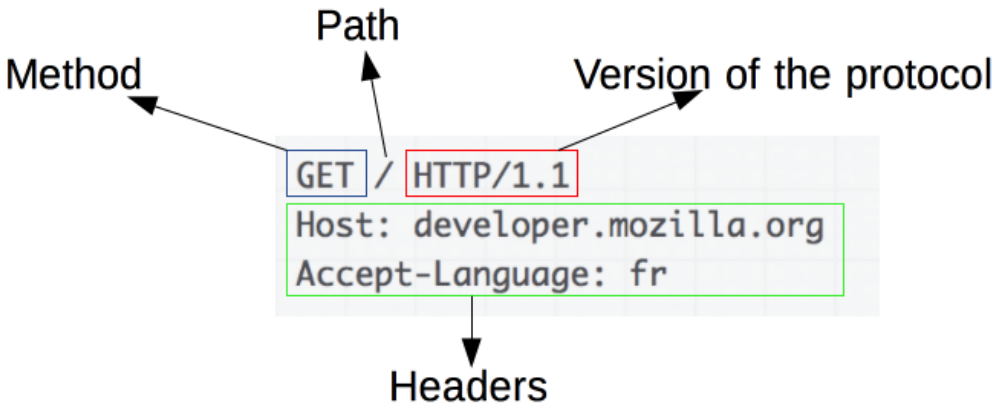


FIGURE 1 – Exemple de requête HTTP

On remarque qu'une très courte chaîne de caractères contient une grande quantité d'informations. Pour nous, la méthode sera toujours la même, car la seule communication que nous voulons effectuer est simplement de récupérer le contenu des sites web, nous n'avons rien à envoyer aux serveurs. On se servira donc seulement de la méthode GET.
Il est aussi possible de rajouter des headers, mais qui, pour notre cas, ne seront pas utiles, la plupart des headers doivent contenir des informations qui nous feraient perdre en optimisation.
On peut illustrer cet exemple avec le header "Date", qui permet d'indiquer dans la requête, l'heure à laquelle celle-ci a été écrite, il faudrait donc obtenir l'heure de la machine pour en faire une version formatée qui, au final, n'apportera rien à la requête.
On retrouve le même cas de figure pour "Expires", qui au lieu d'indiquer l'heure à laquelle la requête a été effectuée, nous indique quand la requête sera "périmée".

6.2.2 Communiquer avec le serveur

Maintenant que la requête est créée, il faut l'envoyer au serveur pour récupérer les informations si précieuses que contiennent Internet.
Pour cela, il a été préférable d'utiliser une librairie qui permet d'envoyer la requête ainsi que de récupérer le résultat. Cette librairie est la librairie "Reqwest" qui comme son nom l'indique, peut traiter des requêtes internet, ainsi, une méthode *send()* nous est fournie et nous permet de communiquer avec le serveur, ce qui nous sera très utile pour récupérer le code source de chacun des sites.

6.2.3 "Seeding" ou ensemencement du crawler

Pour que le crawler fonctionne bien et soit amené à explorer une multitude de variétés de sites, il lui faut partir de sites web pertinents et qui donnent accès à plusieurs sites web qui seront eux aussi capable de reproduire le même processus.
Il faut savoir que ces sites seront la base de la recherche de notre crawler, il faut donc les choisir avec précisions tout en étant certains qu'ils aient le moins de liens possibles entre eux pour éviter que deux sites "source" se retrouvent trop rapidement avec les mêmes références.
On peut se représenter cette situation simplement en partant de deux "sujets" différents mais qui traitent d'un même domaine comme le théorème de Pythagore et le théorème de Thalès autour des mathématiques. Tout en restant sur Wikipédia, on peut, en moins de 5 clics, passer d'une page à l'autre. Ainsi, on comprend que plus deux sujets seront éloignés les uns des autres, moins ils auront de chances de contenir les mêmes hyperliens.
Nous avons donc choisi comme sites web de départ les sites suivants :

- La page wikipédia du nombre 42 en anglais
- Un article scientifique trouvé dans le journal "Le Point" sur les éclipses solaires
- Un article philosophique axé sur la question : "L'argent fait-il le bonheur ?"
- La page web de la coupe du monde de football trouvée sur le site de "L'équipe"
- Une question autour des web crawlers posée sur le site StackOverflow

6.2.4 Expansion du crawler

En ce qui concerne l'expansion du crawler, nous avons décidé de l'implémenter tel un parcours en profondeur de graphe.
L'utilisation des files nous simplifie grandement la tâche, nous l'initialisons comme contenant les liens de nos 5 sites de base. Nous prenons ensuite chaque élément de la file (dans l'ordre) pour y ajouter tous les sites "voisins", et répéter le processus jusqu'à vider la file.
Évidemment, pour des raisons simples de stockage, de patience et de sécurité, nous n'allons pas naviguer dans l'entièreté d'Internet. Nous pouvons définir une limite au crawler, qui, au-delà de cette limite, arrêtera de parcourir les sites web et stoppera son processus. Une sorte de sécurité pour ne pas avoir à visiter les millions de sites web disponibles.
L'avantage d'avoir créé le filtre de Bloom dont nous avons parlé lors de la soutenance précédente est que les expansions vers les sites web déjà visitée n'est pas effectuée, et l'algorithme du filtre du Bloom nous

permet de vérifier cette présence ou non extrêmement rapidement.
Malheureusement, la contrainte à cela est que si un élément est détecté comme déjà visité, nous prenons le risque de ne pas faire de vraie vérification pour s'assurer à 100% qu'il n'a pas déjà été visité car cela prendrait trop de temps, il est donc possible que certains sites ne soient pas visités

6.2.5 Pistes d'optimisation

Avant de débuter cette section, il est important de préciser que pour la partie suivante, le terme "optimisation" ne désigne pas seulement une réduction du temps d'exécution ou une réduction du coût en ressources, mais aussi de la qualité du crawl.

Pour commencer, si on cherche à exporter DroXyd à l'international, on ne pourra pas se contenter d'avoir visité tous les sites web français, il faudra s'ouvrir au monde, et donc, devoir potentiellement changer nos adresses de départ.
Ensuite, certains pages sur Internet possèdent un nombre incalculables de liens externes, quelques fois, ce sont des sites qui existent seulement pour ça, mais nous avons peu de chances de trouver leurs adresses puisqu'ils ne sont référencés sur presque aucun site. Seulement, il existe même des pages Wikipédia qui posent ce problème, notamment la page du nombre 42, en effet, cette page possède un lien externe pour chacuns des nombres compris entre 1 et 1000. Pour contrer ces comportements, la piste d'une sélection aléatoire de sites web en file d'attente pourrait être abordée, on aurait donc une sélection de sites, dans laquelle on pourrait choisir le prochain site à visiter de manière aléatoire, ce qui pourrait aussi permettre un catalogue de domaines encore plus garni.
Une optimisation des ressources pouvant être abordée serait celle du multi-threading, une notion que nous avons étudié autour de la fin du troisième semestre en C. Il s'avère qu'il serait possible d'utiliser cette particularité en Rust, ce qui pourrait grandement réduire le temps d'exécution du crawler.

Pour le moment, toutes ces pistes restent seulement théoriques et nous verrons un peu plus tard si elles pourraient réellement améliorer le crawler.

7 User Input Processing

7.1 SQL query

Dans cette section de notre projet, nous nous sommes concentrés sur l'utilisation de Rust pour exécuter des requêtes SQL afin d'interroger une base de données. Ce processus n'a pas été simple, notamment lors de l'installation des différentes bibliothèques et outils nécessaires à notre tâche, qui fût une tâche très compliquée. Après avoir examiné différentes options, nous avons choisi d'utiliser SQLite et Diesel pour leur simplicité et leur efficacité, car ils nous permettaient de créer des bases de données, de chercher dans celle-ci et même de modifier ces bases de données, les problèmes rencontrés pendant l'utilisation de diesel et SQLite sont le manque de documentation et la prise en main qui est en corrélation avec le fait qu'il n'existe pas beaucoup de documentation, mais une fois ces étape passées, nous pouvions développer un accès à cette base de données.

Un des dilemmes auxquels nous avons été confrontés était de décider de la structure de notre base de données. Devions-nous opter pour une seule base de données avec 2 tables, ou diviser nos données en deux bases distinctes contenant une seule table chacune ? Après mûre réflexion, nous avons décidé que la première option serait la plus utile, simplifiant ainsi notre architecture et notre logique de requêtage.

L'algorithme que nous avons développé pour cette partie de notre projet peut être décomposé en plusieurs étapes.
Tout d'abord, nous établissons une connexion à notre base de données. Pour pouvoir exécuter des requête SQL sur celle-ci. Ensuite, pour chaque élément de la structure que nous souhaitons traiter, nous recherchons les informations correspondantes dans la première table qui comporte deux éléments le premier correspond aux mots-clés présents dans les sites visités, donc, keywords, et le deuxième élément est un ID qui permet de savoir quelle URL prendre.

pizza	1
pizza	2
pizza	3
tacos	4
cat	5
dog	0
cat	6
cat	7
cat	8
dog	0

FIGURE 2 – Table 1

Puis, avec ces informations en main, nous procédons à une recherche dans la deuxième table pour obtenir des données complémentaires ou associées, en utilisant l'ID de l'autre table, ce qui nous permet de récupérer toutes les informations concernant les sites cherchés.

id	url	langue	name	date
Filtre	Filtre	Filtre	Filtre	Filtre
1	https://fr.wikipedia.org/wiki/Pizza	fr	wikipédia	09/04/2024
2	https://www.dominos.fr	fr	dominos	09/04/2024
3	https://www.pizzahut.fr	fr	pizzahut	09/04/2024
4	https://fr.wikipedia.org/wiki/Taco	fr	wikipédia	09/04/2024
5	https://fr.wikipedia.org/wiki/Chat	fr	wikipédia	09/04/2024
6	https://www.larousse.fr/...	fr	larousse	09/04/2024
7	https://www.woopets.fr/chat/	fr	woopets	09/04/2024
8	https://www.santevet.com/...	fr	santevet	09/04/2024
9	https://fr.wikipedia.org/wiki/...	fr	wikipédia	09/04/2024
10	https://www.woopets.fr/chien/	fr	woopets	09/04/2024

FIGURE 3 – Table 2

Puis, une fois que nous avons récupéré les données nécessaires, nous les trions et les filtrons pour ne conserver que celles qui nous intéressent vraiment, en recherchant parmi les résultats de la requête SQL.

Enfin, nous renvoyons ces informations pour les utiliser dans la suite de notre application ou de notre système en utilisant une structure.

Cette approche nous a permis de créer un système efficace pour chercher dans notre base de données en utilisant Rust, offrant ainsi une solution fiable pour nos besoins en matière de gestion de données.

7.2 Suggestions

Le but de cette partie est d'implémenter un pont entre l'utilisateur et le moteur de recherche DroXyd. Il se doit d'être pratique, complet et intuitif. C'est pourquoi ont été implémentées les fonctions de recherche et de correction à une page web qui nous servira d'interface.
Le contenu de cette dernière est divisée en 2 parties. Les améliorations liées aux recherches de corrections, ainsi que le contenu même de l'interface.

Tout d'abord, reprenons les fonctionnalités du dictionnaire tel qu'il l'était à la première soutenance. On pouvait ajouter des mots, les chercher, les compléter ou les corriger. La structure, un arbre ternaire de recherche, est donc la meilleure solution pour stocker et exploiter les données. Seulement, les résultats étaient simples, ils étaient triés par ordre alphabétique et ne fonctionnaient que si le mot était écrit en minuscule et sans caractères spéciaux. Aussi, il ne pouvait pas analyser une phrase en entier, et décerner les mots importants.

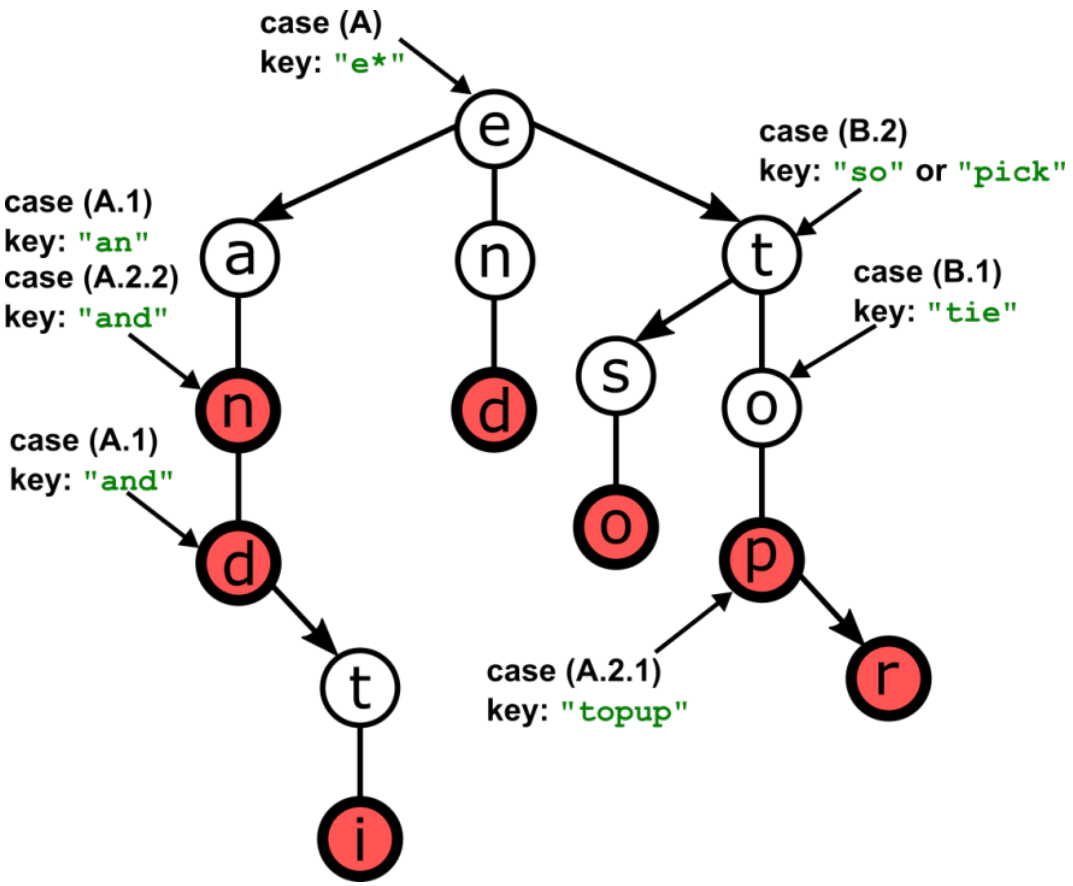


FIGURE 4 – Valuated Ternary Search Tree

En effet, il était primordial d'adapter le dictionnaire à une utilisation fluide de recherche. Il est désormais capable d'analyser une phrase en entier, d'en dégager les mots les plus importants (en les valuant lors de l'ajout d'un mot dans l'arbre), et de corriger intelligemment les termes dans une phrase. Les mots comprenant des caractères spéciaux, comme "-cat" ou "site :wikipedia" sont traités ultérieurement par le parseur, ainsi ils ne sont donc pas à corriger. Il en va de même pour les dates (ou suites de chiffres), qui sont ignorées lors de la correction mais pas lors de la recherche de résultats.

Enfin, les mots mal orthographiés ou incomplet sont revus afin de proposer la meilleure solution. Si le mot peut être complété, alors on prend la suggestion avec la valeur la plus haute sensible de correspondre à l'entrée. Autrement, on choisit la meilleure correction possible, ou dans le pire des cas on supprime l'entrée. Aussi, ont été implémentées les fonctions de suggestions avancées. La requête entière est donc analysée et corrigée, et les résultats de celle-ci sont triés puis affichés. Les mots-clés des résultats les plus courant sont alors mis en avant pour affiner les recherches de l'utilisateur.

Voici quelques exemples :

— On ajoute les mots valués suivants au dictionnaire : ferrari (45), ferry(12), ferraille(3).
— On entre "ferr", et le premier résultat sera alors "ferrari" car il a la valeur la plus élevée (suivi de ferry, puis ferraille)
— On entre "berrari f40 hthhthth", et le résultat suggéré sera "ferrari f40", le premier mot étant corrigé, le second ignoré, et le troisième supprimé. Puisque les mots ferry et ferraille ont une valeur plus basse (car moins présents dans nos bases de données), ils sont moins susceptibles d'être proposés directement. Ils apparaîtront dans les propositions d'auto-complétion les moins pertinentes.

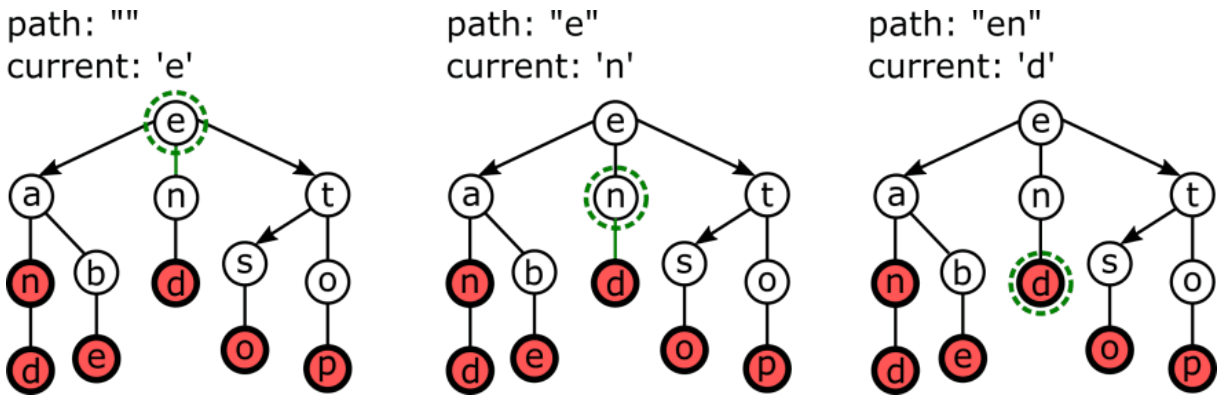


FIGURE 5 – Valuated Ternary Search Tree Example

7.3 Interface

Pour l'interface, il a fallu implémenter une structure pratique afin de simplifier les recherches de l'utilisateur. De ce fait, il est important de créer un environnement propice au développement UI dans le projet. Comptant réaliser l'interface avec Quentin pour la soutenance 3, il est nécessaire de préparer un support flexible au travail en groupe. C'est pourquoi nous avons utilisé roquet, une librairie de développement d'interface web. Dans l'interface DroXyd Search Engine, il est possible de créer plusieurs tabs de recherche, d'entrer une requête et le nombre de résultats voulus, et de voir dans une nouvelle fenêtre le résultat. Ce dernier est constitué de la requête d'origine, de la requête suggérée, et l'ensemble des résultats du dictionnaire sur les mots entrés. Ils seront utiles pour implémenter le menu déroulant d'auto-complétion. Enfin, les résultats de la requête sont affichés, avec la liste des mots clés suggérés. Il s'agit là d'un exemple d'affichage du résultat lors de l'appel à la base de données. La liste de résultats est traitée puis les mots clés sont affichés comme tel. Les requêtes précédentes sont sauvegardées dans le navigateur.

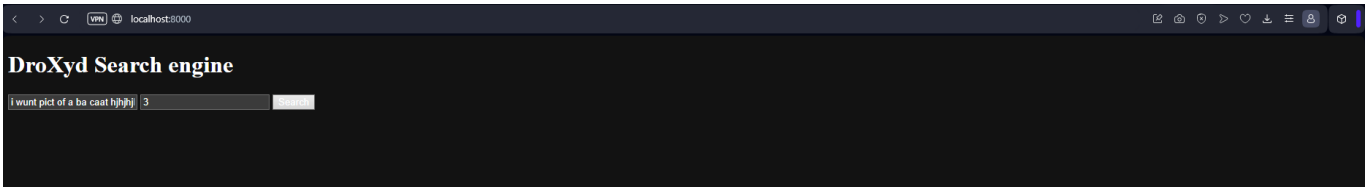


FIGURE 6 – DroXyd Search Engine (Query)

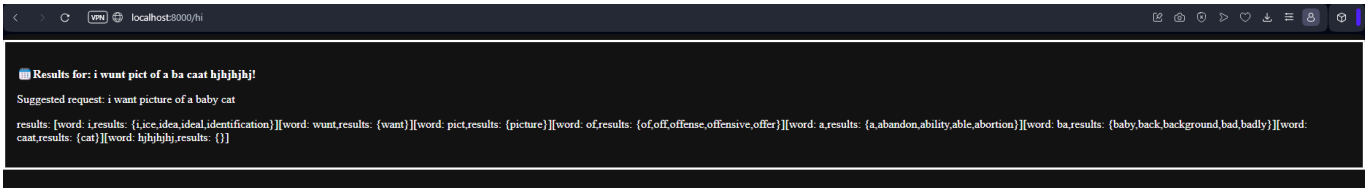


FIGURE 7 – DroXyd Search Engine (Results)

8 Avancement & répartition

Répartitions					
	Léo	Justine	Quentin	Alexandre	Avancement
Crawler	Scalable Bloom Filter				100%
	Seeding + Expanding				100%
	Performance optimization				5%
Scraper		Data Extraction			100%
		Data Cleansing			100%
		Inverted Index			5%
User Input			Parsing		100%
			Lexing		100%
			SQL Queries		100%
				Spell Checker	100%
				Suggesting	100%
				Keyword Filtering	-
Visualizer			Interface	Interface	30%

- $1^{ère}$ soutenance
- $2^{ème}$ soutenance
- $3^{ème}$ soutenance

9 Conclusion

D'un point de vue global, notre projet avance très bien depuis la première soutenance qui s'était déjà très bien passée. Encore une fois, nous avons tous réussi à finir nos tâches tout en gardant une bonne ambiance au sein du groupe. Nous pouvons donc commencer sereinement notre troisième et dernière partie sans avoir une pression à cause d'une partie non terminée. Nous avons même réussi à prendre un peu d'avance sur l'organisation de certaine structure de données comme notre base de données. Notre ressenti général est positif, grâce à l'ambiance mais aussi au projet que nous sommes en train de réaliser. Ce projet, même s'il reste complexe, est aussi passionnant qu'intéressant.

De plus, nous aimerions améliorer la rapidité et l'optimisation de certains de nos algorithmes pour qu'ils soient non seulement plus efficaces mais qu'ils puissent nous donner une précision satisfaisante.

Nous voudrions, une fois le projet terminé, essayer de réaliser quelques tâches en plus si cela est possible, telles que faire une barre de favoris, implémenter des petits jeux ou pouvoir mettre des thèmes customisés. Nous restons confiants pour la suite du projet, et nous avons hâte de le mener à bien.

10 Bibliographie

Crawler

https://fr.wikipedia.org/wiki/Hypertext-Transfer-Protocol
https://crates.io/crates/reqwest

Liens de semencement du crawler :

https://en.wikipedia.org/wiki/42$_{(number)}$
https://www.lepoint.fr/eureka/qu-est-ce-qu-une-eclipse-solaire-totale-08-04-2024-2557109$_4$706.php
https://www.psychologue.net/articles/largent-fait-il-le-bonheur
https://www.lequipe.fr/Football/Coupe-du-monde/
https://stackoverflow.com/questions/tagged/web-crawler

Scraper

https://users.rust-lang.org/t/how-to-check-for-internet-connection/89893/5
https://hyper.rs/
https://stackoverflow.com/questions/150750/hashset-vs-list-performance
https://stackoverflow.com/questions/.
https://users.rust-lang.org/t/in-memory-btreemap-like-crate-that-supports-acid-transactions/106457/2
https://www.dotnetperls.com/btreemap-rust
https://iq.opengenus.org/hashmap-and-btreemap-rust/
https://www.reddit.com/btreemap-v-hashmap
https://youtu.be/vZAXpvHhQow?si=6QycA6u2b3DBkSQy
https://youtu.be/OymqCnh-APA?si=IHNMOUhmKGko3x3E
https://youtu.be/D2V1okCEsiE?si=N-VTsyA2fgM5Zmpa
https://programminghistorian.org/analyzing-documents-with-tfidf
https://users.rust-lang.org/t/watch-out-for-nans/70016/5
https://datascientest.com/tf-idf-intelligence-artificielle
https://docs.rs/reqwest/latest/reqwest/