Rapport de soutenance 1

Quentin Couavoux
Léo Menaldo
Alexandre Colliard
Justine Gueulet

Table des matières

1 Plan de soutenance

Introduction

Démonstration

- User Input Processing
- Crawler
- Scraper

Conclusion

- Éléments terminés
- Éléments commencés
- Éléments à commencer

2 Prélude

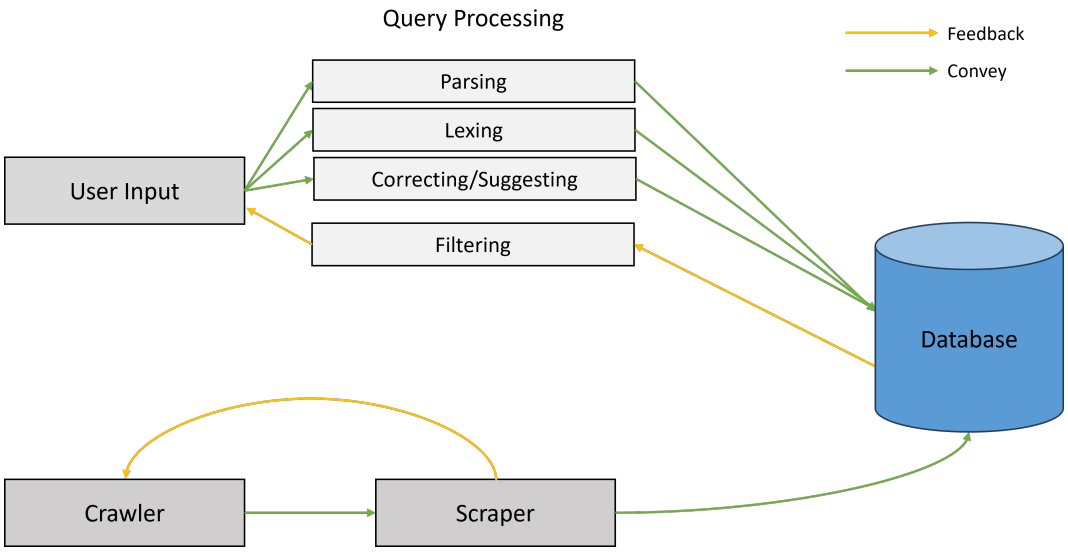
2.1 Le projet

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

Dans ma partie de cette première soutenance, j'ai pris en charge le développement d'un analyseur lexical et d'un parser pour traiter les saisies de l'utilisateur. En combinant ces deux composantes, nous avons pu créer un système capable de traiter une variété de requêtes utilisateur.

4.2 Léo Menaldo

Le filtre de Bloom doit être implémenté. Ce filtre utilise un algorithme probabiliste dont nous parlerons plus tard.

4.3 Alexandre Colliard

Dans la partie "User Input", je devais implémenter un système capable de détecter l'existence de mots dans un dictionnaire ou d'apporter d'éventuelles suggestions et corrections. Le but est d'améliorer la précision des recherches et d'accompagner l'utilisateur lorsqu'il fait ses recherches.

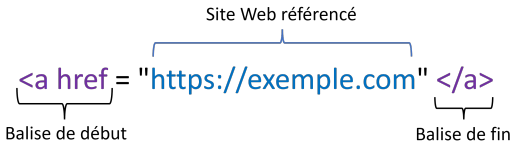
4.4 Justine Gueulet

Le scraper se charge d'extraire les données contenues dans le code source HTML d'un site Web. Il est capable de renvoyer la langue détectée, ainsi que les URLs et les paragraphes.

5 Extraction des données

5.1 URLs

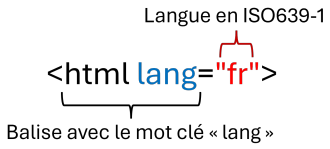
Une pratique commune lors de la création d'un site web est le référencement de sources, généralement des sites web, laissant à un utilisateur la possibilité de découvrir de nouveaux sites. Pour accomplir cela, il existe en langage HTML des balises. Ces dernières sont utilisées pour formater, structurer et hiérarchiser les données tout en permettant au navigateur de savoir comment afficher les informations.
Pour référencer un site web dans un code source HTML, il est nécessaire d'utiliser la balise "<a href>" suivie d'un lien valide vers une autre page web.



Il est également possible d'utiliser ces balises pour faire remonter l'utilisateur sur une partie spécifique de la page web ou l'amener sur une autre section du site web. C'est pour cela que j'ai décidé de chercher les emplacements de ces balises référençant d'autres sites web et de les ajouter à un vecteur au fur et à mesure de mon parcours dans le code source HTML du site web, pour renvoyer un array possédant tous les liens rencontrés dans le code source. Or je me suis vite rendue compte que certaines URLs que j'ajoutais dans mon vecteur ne menaient pas vers d'autres sites. pour parer à ce problème, j'ai décidé d'implémenter pour le moment une fonction vérifiant le format des liens récupérés. Or pour les prochaines soutenances je voudrais faire en sorte de n'avoir que des URLs valides et utilisables afin de pouvoir envoyer directement ces URLs à la partie concernant le crawler. Pour ce faire je vais devoir me renseigner sur les balises de référencement ainsi que sur les différentes version du langage HTML et d'autres langages utilisés tels que le XHTML.

5.2 Langue

Lorsqu'un utilisateur effectue une recherche, le navigateur va devoir détecter plusieurs critères : Certains mots clés, le PageRank d'un site web, . . .Mais la langue est aussi un de ces critères et elle joue un rôle important dans la qualité des résultats renvoyés à l'utilisateur. Pour renseigner la langue, une balise "<html lang="fr">" est utilisée en début de code.



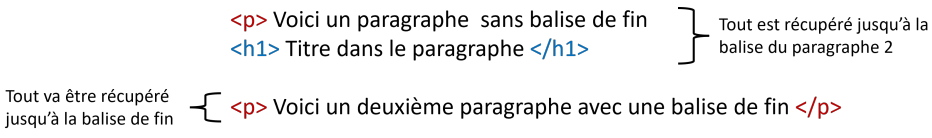
Il faut donc rechercher le mot clé "lang" pour obtenir la langue. Or la langue doit être encodé en ISO639-1, il faut donc effectuer une vérification de la langue. Pour ce faire, il faut regarder si cette dernière fait partie des 217 abréviations de langues.

5.3 Paragraphes

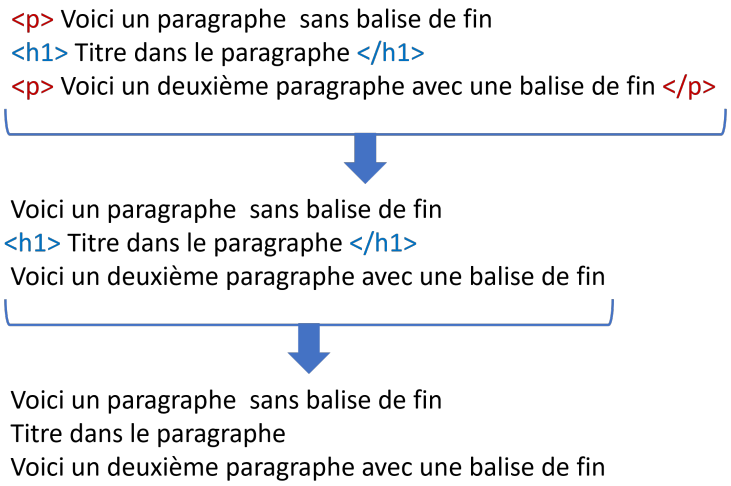
Contrairement aux liens référencés ou que la langue utilisée où il existe une balise particulière pour indiquer au compilateur leur nature, les textes sont plus rudes à gérer. Certains langages de programmation balisés possèdent des conventions qui divergent des autres, ce qui peut rendre la gestion de tous les cas distincts difficile. De manière générale, pour indiquer le début d'un paragraphe, une balise "<p>" est utilisée. Cette dernière signale au compilateur qu'il doit formater d'une certaine façon les données contenu dans cette balise afin d'avoir un rendu concis et clair. Sans cette balise, le navigateur risque de mal mettre en place le texte et peut créer des décalages voire une disparition du texte. Or la balise de paragraphe <p> n'est pas l'unique balise indiquant un texte, il existe également :

— $< DL >$ indique le début d'une liste de descriptions et peut comporter d'autres balises de mise en page.
— $< H_i >$ avec $i \in [\![1;6]\!]$ représente un titre, le chiffre indiquant sa taille.
— $< TT >$ (obsolète) permet d'afficher un élément à largeur fixe tel qu'un téléscripteur.

Les langages de programmation à formatage possèdent une quantité assez surprenante de balises de mise en page, et il faut que nos algorithmes puissent toutes les détecter ainsi que gérer tous les cas possibles. Il faut donc, pendant l'analyse du code source, prendre en compte chacune de ces balises bien spécifiques puis les retirer du paragraphe afin d'avoir un texte propre.
Or, certaines de ces balises sont considérées comme obsolètes et les sites possédant ces dernières courent le risque d'être supprimés. Malgré ce risque, sachant que ces sites sont encore en ligne, il faut tout de même les analyser et gérer le cas des balises obsolètes. Il existe aussi la possibilité de mettre certaines balises de mise en forme à l'intérieur d'autres balises de formatage. On peut par exemple mettre une balise de titre dans une balise de paragraphe. Mais la plus grosse difficulté rencontrée durant l'extraction de paragraphes a été de prendre en compte les différentes versions de langages de programmation de balisage. En utilisant le XHTML, il est obligatoire de fermer une balise de paragraphe <p> avec < /p>, sinon une erreur de compilation risque d'apparaître. Tandis qu'en HTML5 il n'y a aucune obligation d'utilisation une balise fermante pour indiquer la fin d'un paragraphe. Pendant la compilation du code source d'un site Web, la fin d'un paragraphe est indiquée par le commencement d'un autre paragraphe ou avec d'autres balises. Sachant qu'il pourrait être encombrant de créer plusieurs fonctions gérant ces cas séparément, il est plus pratique et simple pour nous de ne se focaliser que sur une unique fonction, surtout que le XHTML est assez peu utilisé de nos jours. Nous commençons par récupérer les emplacements des balises <p>, puis, dans le cas du XHTML il faut également récupérer les emplacements de balises de fin. Nous récupérons ensuite le texte entre les balises. Or dans le cas du HTML, sachant que la personne n'est pas obligée d'utiliser les marquages de fin, nous vérifions tout de même la présence de balises de sortie, s'il en existe, nous récupérons ce qui est écrit entre les deux balises, sinon nous interceptons tout ce qui est écrit jusqu'au début de la prochaine balise.



Une fois les balises de paragraphe gérées, nous obtenons en tant que résultat une chaîne de caractères contenant le texte voulu. Or il possède des parasites : les autres balises de mise en page n'ont pas été traitées et sont donc encore dans le paragraphe. Nous devons donc les retirer afin d'obtenir un paragraphe compréhensible et clair. Pour ce faire, nous avons regardé de plus près certains sites et nous avons remarqué que les balises indiquant un titre font parties des balises de mise en forme les plus utilisées après celles de mise en page de paragraphe. Les balises de titre sont assez compliquée à gérer car dans certains langages de programmation balisés en possèdent 6 (comme le HTML5). Nous avons donc créé un tableau contenant ces six formulations possibles pour ensuite les comparer aux balises restantes dans mon texte. Pour nous éviter de parcourir tout le paragraphe, nous recherchons tous les emplacements du caractère '<' qui énonce généralement le 1^{er} caractère de la formulation d'une balise. Ce processus est répété avec tout autre type de balise communément utilisés.



Nous cherchons une façon d'optimiser certaines fonctionnalités afin d'éviter de devoir parcourir plusieurs fois l'entièreté du paragraphe, or nous n'avons pas encore trouvé d'alternatives pour le moment.

5.4 Recherche de mots clés

Maintenant que nous arrivons à extraire les informations importantes du code source d'un site web, autrement dit la partie du "Data Extraction", nous avons décidé de prendre de l'avance et de nous renseigner sur cette prochaine partie qu'est le "Data Cleansing". Une fonctionnalité très importante et complexe du "Data Cleansing" est la récupération de mots-clés d'un texte. Pour ce faire, l'algorithme du TF-IDF joue un rôle crucial. L'algorithme du TF-IDF (abréviation pour "Term Frequency-Inverse Document Frequency") marche en utilisant des statistiques de pondération indiquant l'importance d'un mot, comme son nombre d'occurrence par exemple, dans un texte particulier.

6 Le filtre de Bloom

Le filtre de Bloom et son algorithme forment un outil extrêmement puissant basé sur la probabilité de présence d'un élément dans un ensemble, voyons en détail son fonctionnement et son utilisation dans notre projet.

6.1 Fonctionnement

Le filtre de Bloom en lui-même n'est qu'une simple liste composée de valeurs égales à 0 ou 1. Premièrement, l'entièreté du filtre est initialisée à 0. Soient 3 fonctions de hachage H_1, H_2 et H_3 que nous définirons plus tard, ainsi que x, un mot que l'on souhaite ajouter à l'ensemble. C'est à l'aide de ces 3 fonctions que nous allons ajouter un élément à notre ensemble.
En supposant que H_1, H_2 et $H_3 : E \longrightarrow$ [0, Taille de l'ensemble]. Alors, nous allons modifier les valeurs situées aux emplacements $H_1(x)$, $H_2(x)$ et $H_3(x)$ du filtre par 1.
L'objectif étant d'appliquer ce processus à tous les éléments que nous comptons faire appartenir à l'ensemble.
La question qui se pose naturellement est donc : Comment savoir si l'élément recherché est présent dans l'ensemble ?
Pour cela, il va falloir appliquer nos 3 fonctions de hachage au mot y que l'on recherche. Les valeurs situées aux emplacements $H_1(y)$, $H_2(y)$ et $H_3(y)$ seront donc égales à 0 ou 1. Si au moins l'une d'entre elles vaut 0, on pourra alors affirmer que l'élément ne fait pas partie de l'ensemble.
En revanche, il est possible que les 3 valeurs valent 1. Mais ce n'est pas pour autant qu'on peut affirmer à 100% que l'élément est présent, en effet, il existe un risque d'obtenir un "Faux négatif", cet évènement peut s'expliquer par le fait que d'autres mots peuvent avoir les mêmes valeurs de hachage.
Dans ce cas, une des solutions les plus efficaces pour remédier à ce problème est d'augmenter la taille du filtre, ce qui peut nous permettre d'éviter certaines collisions. Évidemment il est possible que d'autres collisions apparaissent suite à ce changement mais on partira du principe que la probabilité que ce cas se présente est extrêmement faible.

6.2 Fonctions de hachage

Nous avons précédemment parlé de fonctions de hachage, pour l'implémentation du filtre de Bloom, nous en utiliserons toujours 3 et ce choix est justifié. Utiliser moins de 3 fonctions de hachage résulte à une fréquence beaucoup trop élevée de résultats "Faux positifs" donnés, puis, en utiliser plus de 3 nous amène à un nombre de collisions trop élevé, et pour éviter ces collisions, il faudrait augmenter la taille du filtre, qui, en se projetant dans le futur, sera déjà assez grand comme cela.
Parlons maintenant un peu plus en détail de ces 3 fonctions :

- Nous définirons $H_1(x)$ comme la fonction de hachage cryptographique "MD_5", autrefois utilisé pour calculer l'empreinte numérique d'un fichier, elle s'avère en réalité très utile pour simplement hacher un mot ou une chaîne de caractères, au vu de la complexité du résultat et du fait que le résultat soit encodé sur 128 bits.
- $H_2(x)$ comme la fonction "SHA-256", cette fonction fait partie d'une "famille" de fonctions de hachage conçue par la NSA, contenant aussi la fonction "SHA-512", celles-ci ont été beaucoup inspirées du modèle de la fonction MD_5. Cette fonction, quant à elle, renvoie un résultat encodé sur 256 bits.

La différence flagrante que l'on trouve entre H_1 et H_2 est la taille du résultat donné par les fonctions (128 contre 256). Mais elles possèdent toutes deux un point commun qui les démarque des simples fonctions de hachage.

```
SHA-256 Tests
DroXyd is the best project made by 2027 students ! = c23132328bf20f1fd17532ad134f12644750f3f2570b32b56d2f6773719d8b4
Droxyd is the best project made by 2027 students ! = af4e0c57b7af23e3a42dd1693893644bc88567cd70c8612bb77bf6c1380fd4e
DroXyd is the best project made by 2028 students ! = ace1b60e9dc4838723967cdc733db5b3576a7a7427957c1de307eb14d66b809
droXyd is the best project made by 2027 students ! = 7d9db59d3b147c7f1f6f74f4be5413be1125a1be2041c840cb7f93ef15c66f7c
droxyd is the best project made by 2027 students ! = 281bcae044c34723e228e05778ef72dafa55f1a96c91d6265eb472bf961e93b1
DroXyd is the best project made by 2028 students !! = e65993a7da42f71c9a5dff76f256336a19a554fd792537f2e6e66dceb468cb1d
```

FIGURE 1 – SHA-256 : Plusieurs tests réalisés sur des strings presque similaires

```
MD5 Tests
DroXyd is the best project made by 2027 students ! = b6d843baf95d6842544a99b72671112f
Droxyd is the best project made by 2027 students ! = 2ad15bbacd5680422843b1b7c66a292f
DroXyd is the best project made by 2028 students ! = 778073bae30598421df2c9b7c819412f
droXyd is the best project made by 2027 students ! = b6d843baf95d6842544a99b72671112f
droxyd is the best project made by 2027 students ! = 2ad15bbacd5680422843b1b7c66a292f
DroXyd is the best project made by 2028 students !! = 7f06f3ba288c1842637949b7d9fc12f
```

FIGURE 2 – MD_5 : Plusieurs tests réalisés sur des strings presque similaires

— Finalement $H_3(x)$ sera définie comme étant équivalente à $H_2(H_2(x))$, cette fonction est donc une sorte de (SHA-256)^2, qui donne des résultats très convaincants. Cette fonction renverra un résultat encodé sur 256 bits elle aussi, et ce résultat sera très différent de celui renvoyé par H_2.

6.3 Exemple

Posons notre ensemble de départ comme étant composé des mots suivants :
Mickey, Minnie, Donald, Daisy, Dingo et Pluto !

```
List of words :
md5     Mickey: 304483da63d2dc62a2c00dd7a0fa854f
sha     Mickey: 86213dc58f8799d67eb40f3787b267a1a7e04d8d3f0599058acbd767b4eb1c0c
shasha  Mickey: c68bb855e196d69f3445fe30f1c254deeca9c64d2e5f21a78f90d794c7cfd9b

md5     Minnie: 67cb5e1a46fcb6a285e9e8171c845f8f
sha     Minnie: 7d82aef8cd8ebaf7fbc24e199f3d2443937906e428077c14b8b19bf5f30ba1f
shasha  Minnie: cbd28642b4105f88917806b746fd2efd77fe018b7e5f7f6d70d8af705df6cfb4

md5     Donald: febcfc5ad8cb54e217b886576df2fdcf
sha     Donald: 67c01096586bfee3c8925f548c476cf6b6fcc66a15a959cf4c3d3b18d97665ca
shasha  Donald: c67ad66c672f8a7416dda42df8675547412d7f9b28e2247b79358fc99403a08

md5     Daisy: 9ea529da7db38262bca0b3d72adb2b4f
sha     Daisy: 936b0e80932a8774152780cbce3c2329ce29424f50ee4be6a55170c88be8fa7
shasha  Daisy: 37b2add839b6db3ba20fe3d584b7a660ac84d76354f6ab28af95ea904fdc45b3

md5     Dingo: 49e5a0da8173f962c0612ad7d69ba24f
sha     Dingo: 6c2a7712c76a44c6ee982d4719f79d03904dee35cfed7e9f3b098eb98fc53816
shasha  Dingo: add2e068fd9a7c041245cf2497db4248d0b8e3d6463a7830221726acb9d0fbd

md5     Pluto: 2e03129a9f726b22de5f9c979cba140f
sha     Pluto: 327f41ebe7a56725c0262e6529d3793fed31eabf96facbe583690c97be27d
shasha  Pluto: 19afaf564bcc74991c2807f69305b7b399acab6bfbb1a6475842035a2ff27d90
```

FIGURE 3 – Application de chacune des fonctions sur chacun des mots

Après insertion de tous ces éléments, le filtre de Bloom sera mis à jour.

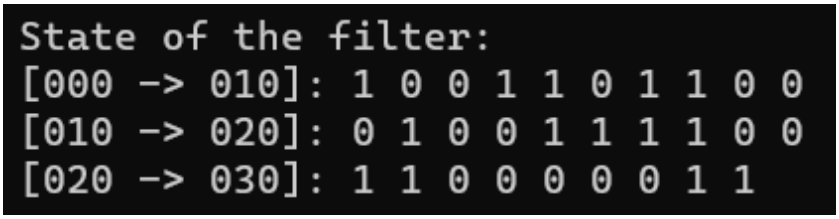


FIGURE 4 – État du filtre après avoir y avoir ajouté tous les mots

La première remarque que l'on pourrait se faire serait d'apercevoir qu'il y a au total 14 / 29 emplacements du filtre occupés par des 1, alors qu'il y avait au total 6 mots à hacher par 3 fonctions chacun, on compte donc parmi tous nos mots des mots qui possèdent des valeurs de hachage égales, ce qui nous arrange car moins on trouve de valeurs fixées à 1 dans le filtre, plus on sera précis !
Il nous reste maintenant plus qu'à regarder si les tests de présence sont vérifiés !

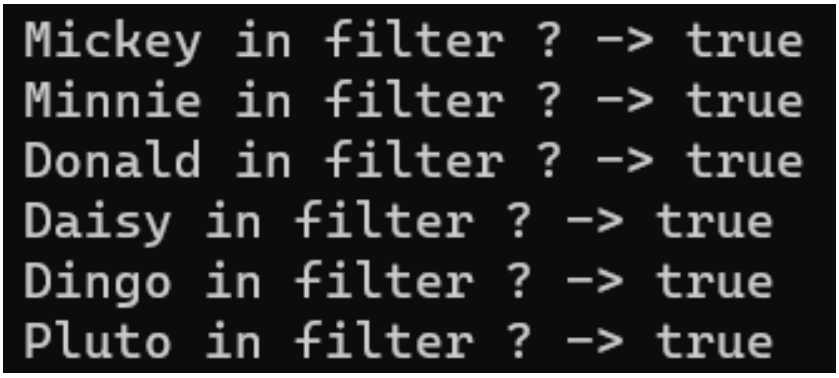


FIGURE 5 – Vérification des mots présents dans l'ensemble de départ

Puis, nous allons pimenter les tests de présence en comptant des mots qui ne sont pas présents dans l'ensemble de départ

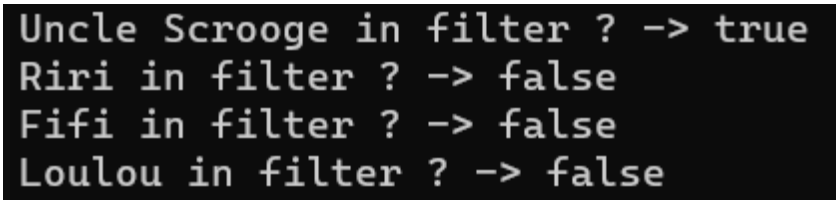


FIGURE 6 – Vérification de mots non présents dans l'ensemble de départ

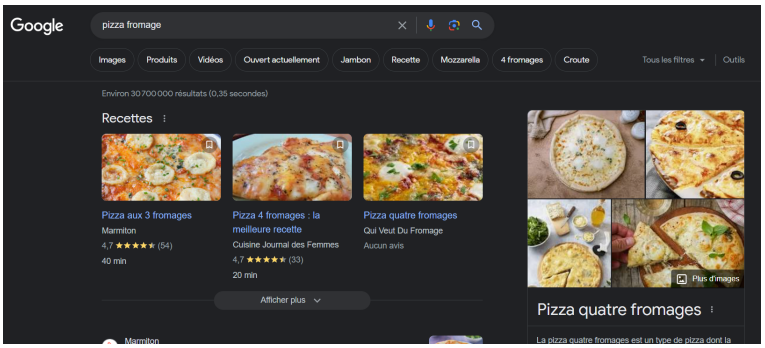
Il était évident que Riri, Fifi, et Loulou ne faisaient pas partie de l'ensemble de départ, et le filtre l'a bien compris. En revanche, il n'en est pas de même pour l'Oncle Picsou ! Qui est quant à lui indiqué comme présent dans l'ensemble, c'est donc un cas de "Faux négatif".
Il faut tout de même garder à l'esprit que le filtre de Bloom nous permet d'éviter le plus possible de faire des recherches inutiles dans notre base de données, mais il existe tout de même des cas qui nous obligeront à nous assurer que l'élément n'est pas dans l'ensemble dans le cas d'un "Faux négatif".

7 User Input Processing

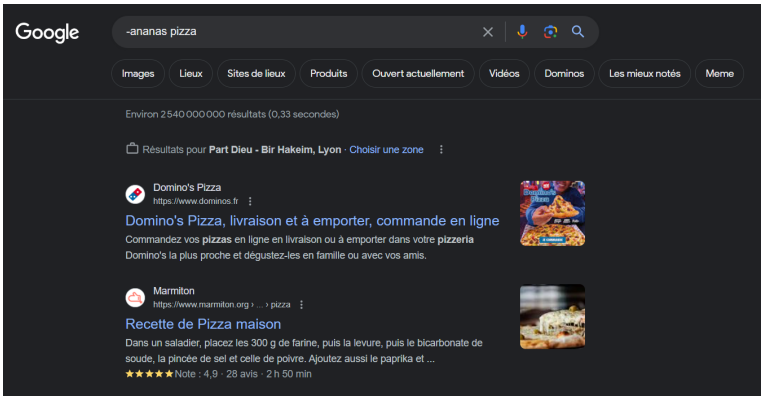
7.1 Lexer/parser

Dans cette partie du projet, nous nous sommes concentrés sur la création d'un analyseur lexical (lexer) et d'un parseur (parser). Ces deux composantes jouent un rôle crucial dans le traitement des requêtes de l'utilisateur. L'objectif principal d'un analyseur lexical et d'un parseur est de comprendre la structure d'un texte ou d'une phrase donnée, en identifiant et en extrayant les éléments clés qui la composent. Dans le cadre spécifique de notre lexer/parser, notre objectif est de comprendre précisément ce que l'utilisateur recherche. Pour ce faire, nous avons défini plusieurs mots-clés qui permettent à l'utilisateur de spécifier sa requête de manière plus précise et efficace.

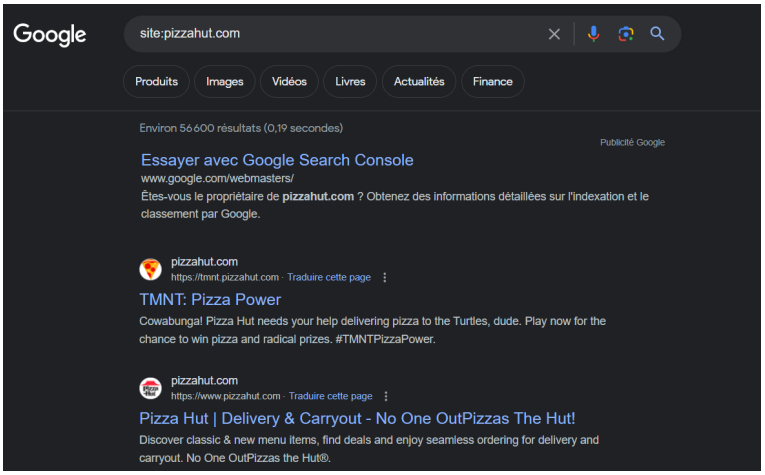
Le premier mot-clé est la possibilité de ne rien spécifier, ce qui correspond à une requête normale. Par exemple, une requête comme "pizza fromage" permettrait de trouver tous les sites contenant les mots "pizza" et "fromage" dans leur contenu.



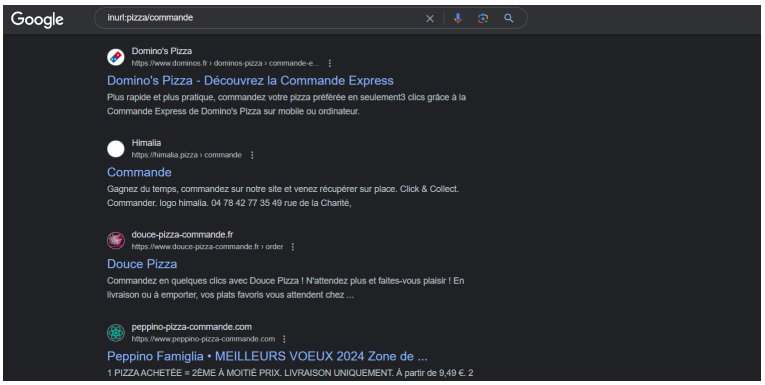
Un autre mot-clé important est le symbole "-", qui est utilisé pour exclure certains termes de la recherche. Par exemple, en saisissant "-ananas", l'utilisateur exclue tous les sites contenant le mot "ananas" dans leur contenu.



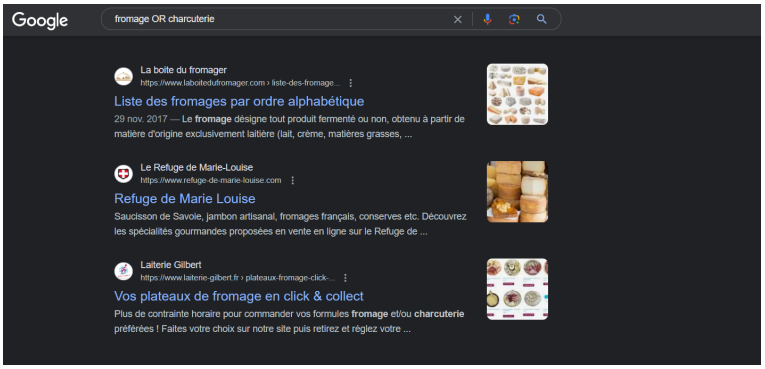
Le mot-clé "site :" est également très utile, car il permet à l'utilisateur de restreindre sa recherche à des sites web spécifiques. Par exemple, en saisissant "site : pizzahut.com", l'utilisateur obtiendrait uniquement les résultats provenant du site pizzahut.com.



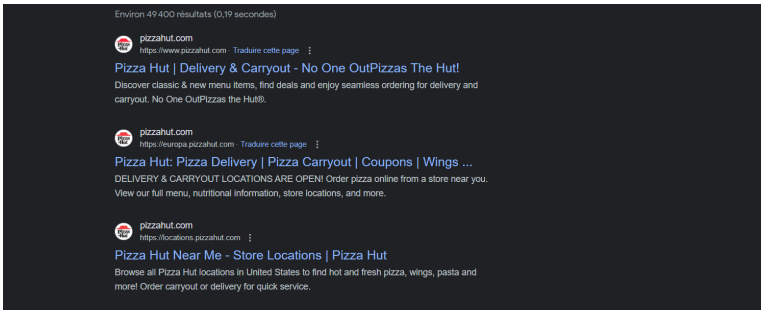
De même, le mot-clé "inurl :" permet de rechercher des sites qui contiennent un terme spécifique dans leur URL. Par exemple, en saisissant "inurl : pizza/commande", l'utilisateur obtiendrait tous les sites dont l'URL contient "pizza/commande".



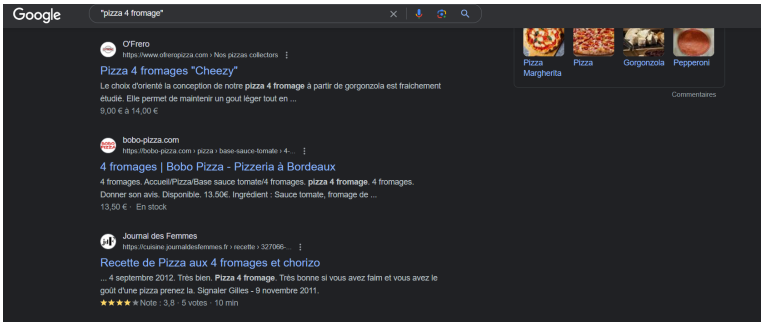
L'utilisation du mot-clé "OR" permet à l'utilisateur d'exprimer plusieurs options dans sa requête. Par exemple, "OR : Fromage/Charcuterie" retournerait des résultats liés à "fromage" ou à "charcuterie".



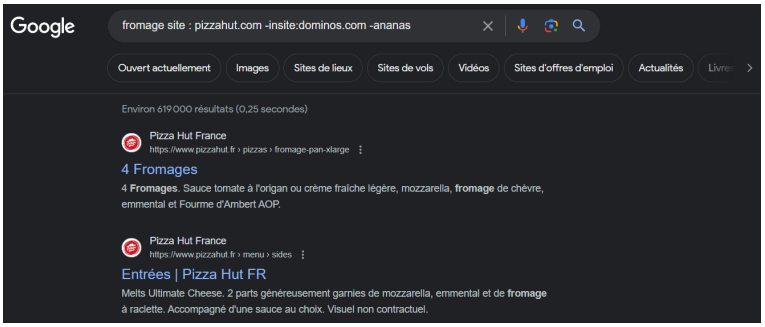
Le mot-clé "unsite" est pratique pour exclure des sites spécifiques de la recherche. Par exemple, en saisissant "unsite : legume.com", l'utilisateur ne verrait pas de résultats provenant du site légume.com.



Enfin, l'utilisation de guillemets (« ») permet à l'utilisateur de rechercher une phrase exacte dans les résultats. Par exemple, « pizza 4 fromage » retournerait des sites contenant exactement cette phrase.



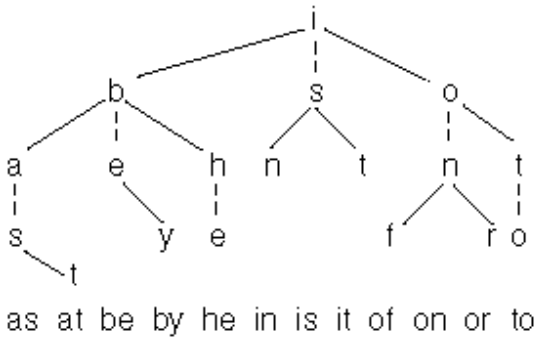
En combinant ces différents mots-clés, l'utilisateur peut affiner sa recherche pour obtenir des résultats plus pertinents. Par exemple, en saisissant "fromage site : pizzahut.com unsite : dominos.com -ananas", l'utilisateur exclurait les sites de dominos.com, rechercherait spécifiquement sur pizzahut.com et exclurait tous les résultats contenant le mot "ananas".



Ces fonctionnalités offrent à l'utilisateur la possibilité de mener des recherches précises et ciblées, lui permettant ainsi d'obtenir les résultats les plus pertinents en fonction de ses besoins spécifiques.

7.2 Dictionnaire/TST

Pour implémenter des opérations d'orthographe, il est nécessaire de construire un dictionnaire. Le format de ce dernier se doit d'être adapté aux dites opérations pour permettre une meilleure qualité d'exécution, tant en termes de temps et de mémoire. La méthode la plus simple et intuitive serait celle d'une liste pour stocker les mots triés, uns à uns. Pour une base de donnés de 1 000 mots, soit environ 5 000 caractères, il faut compter 10 ko. Afin de chercher un mot, il faudrait une complexité de $o(n)$, tout comme trouver la liste de suggestions. Ajouter un mot demanderait $o(n)$ aussi, puisqu'il faut le placer au bon endroit. Un moyen plus optimisé pour implémenter un dictionnaire sera via un TST (Ternary Search Tree). Il permettrait de réduire la mémoire à environ $o(\log_3(n))$, et chaque opération à $o(\log(n))$.

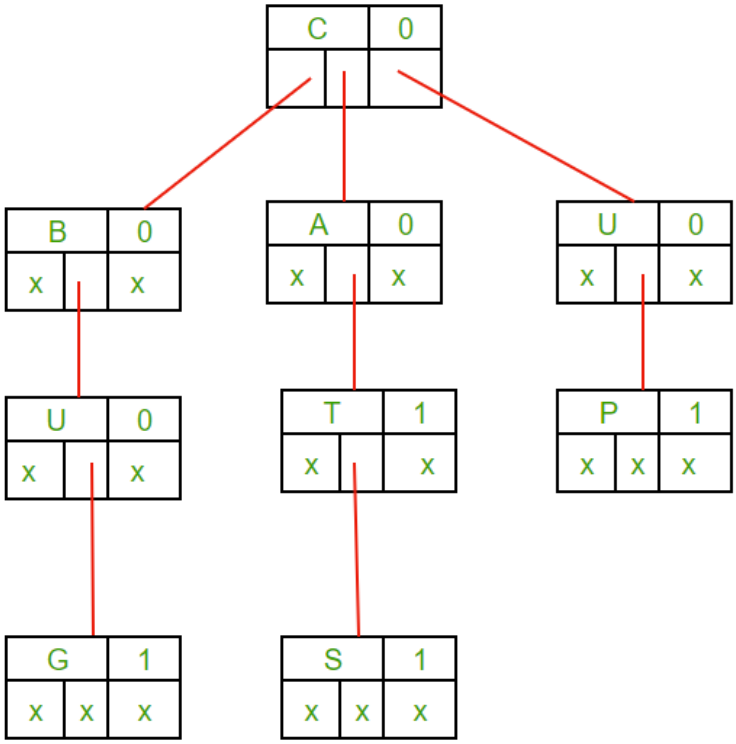


L'implémentation du TST se fait grâce à une liste composée des éléments suivants pour chaque noeud : un identifiant, sa valeur, le fait qu'il représente ou non la fin d'un mot, et 3 pointeurs. Le pointeur dit gauche représente une noeud de valeur inférieure à celle de sa racine, et celle de droite un noeud de valeur supérieure à celle de sa racine. Lors du parcours de l'arbre, les noeuds situées à droite et à gauche sont considérées comme au "même niveau" que la racine. Enfin, le pointeur du centre, noté "équivalent" représente une descente après croisement de valeur recherchée. Par exemple, je cherche l'existence du mot "as" dans l'arbre ci-dessus. La lettre 'a' est inférieure à 'i', donc je descend à gauche. Même opération pour 'b'. Arrivant à 'a', je descend (car même valeur), ainsi je compare 's' avec la valeur du $2_{ème}$ caractère du mot recherché. 's' représenté par ce noeud est considéré comme la fin d'un mot, donc "as" existe dans ce dictionnaire.

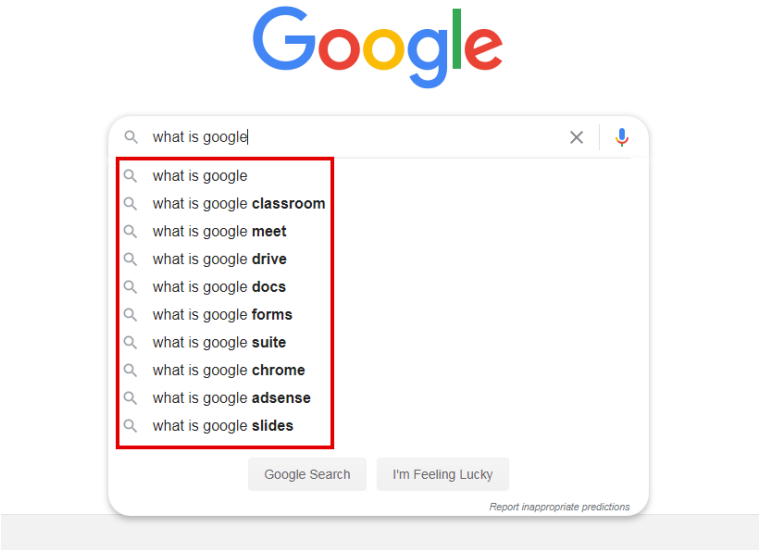
.

.

. Les premières opérations ont donc été implémentées, comme ajouter un mot, rechercher un mot, lister les mots. . .Ce qui permet ainsi de manipuler le dictionnaire et ses composantes. L'objet de cette architecture est de permettre l'algorithme suivant : chercher le mot de l'utilisateur, le compléter si il n'existe pas, ou le corriger si aucune suggestion n'est trouvée. La suite de cette présentation va traiter des algorithmes de parcours et de corrections.



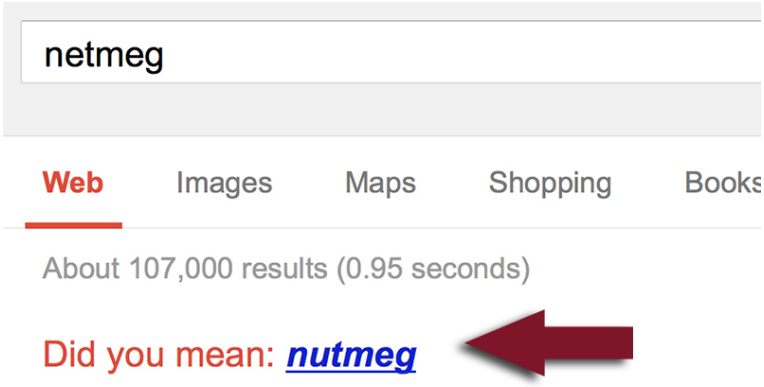
Pour l'autocomplétion, il suffit de suivre l'écriture du mot. Après l'algorithme de recherche d'un mot, on tombe sur un noeud qui n'est pas terminal. L'opération est donc de descendre au noeud "équivalent", puis de lister tout les mots accessibles depuis ce nouveau noeud. La qualité de la suggestion ne dépend que de son emplacement dans le dictionnaire. C'est pourquoi, dès l'accès à la base de données des mots-clefs des sites internet, il sera possible d'orienter ces suggestions aux plus récurrentes.



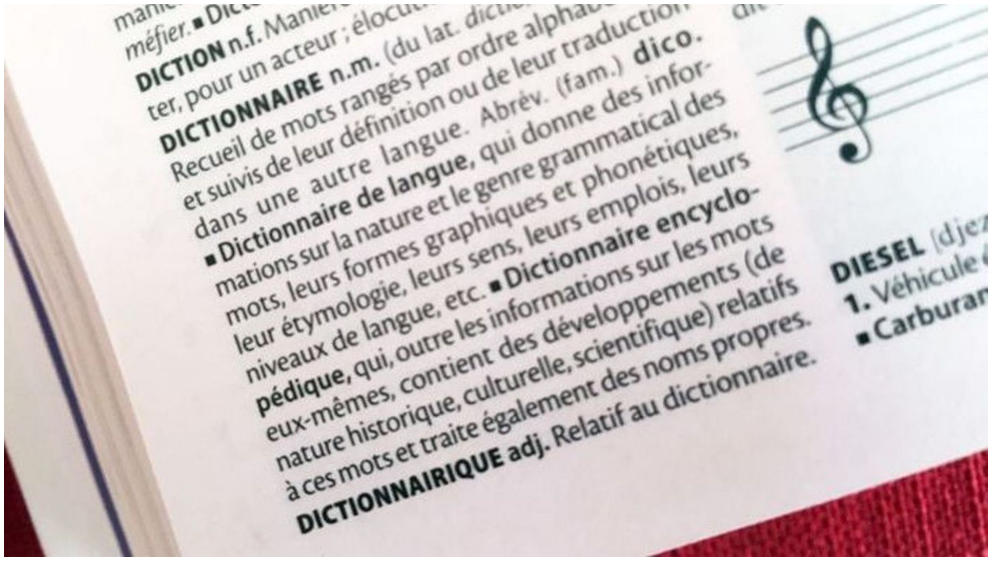
Pour la correction, il faut appliquer des opérations variées sur les mots entrés. Les opérations les plus courantes sont : l'interversion, la suppression, la mutation et l'ajout . Prenons le mot "acb", on obtient les mots suivants :

— interversion : "cab","abc" (Dans le cas où l'utilisateur a appuyé sur les touches simultanément)
— suppression : "cb","ab","ac" (Dans le cas où l'utilisateur a appuyé sur un touche en trop)
— mutation : "bcb","ccb","dcb","ecb"..."aab","abb"... (Dans le cas où l'utilisateur aurait mal tapé une lettre)
— ajout : "aabc","bacb","cacb"..."abbc","acbc"... (Dans le cas où une lettre à été oubliée)

Toutes ces alternatives vont être cherchées dans le dictionnaire. Si un mot est trouvé, on le présente comme "correction". Ces corrections ont une complexité de $o(54n+25)$ avec n le nombre de lettres du mot à tester. Les corrections sont dites à une distance de 1, car une seule faute par mot est autorisée.



Le système de correction est certes fonctionnel, mais peut faire l'objet de quelques améliorations. Lors de l'accès à la base de données de mots-clefs des sites internet, il sera possible de suggérer des corrections par récurrences des mots. Cette amélioration sera mise en place plus tard dans le projet et se nomme "Peter Norvig's Approach". Le choix de la correction de distance de 1 se doit à la quantité de calculs que demanderait la distance de 2 $(o(54n+25)^2)$. De plus, corriger un mot à 2 lettre impliquerait nécessairement la suggestion de tout les mots de 2 lettres du dictionnaire, ce qui est peu précis.
Ainsi, l'application est capable aujourd'hui d'assimiler un dictionnaire entier (sous forme de .txt, un mot par ligne) et de le formater sous forme de TST. Ensuite, il est capable d'analyser une suite de mots et de déterminer si ils existent dans le dictionnaire. Si ce n'est pas le cas, il essayera d'autocompléter ou corriger au besoin.



8 Avancement & répartition

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

9 Conclusion

D'un point de vue global, notre première partie du projet s'est très bien passée, nous avons tous réussi à finir nos tâches tout en gardant une bonne ambiance au sein du groupe. Nous pouvons donc commencer sereinement notre deuxième partie. 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 très passionnant ainsi qu'intéressant.

De plus, nous aimerions améliorer la polyvalence et l'adaptabilité 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

Scalable Bloom Filter

https://systemdesign.one/bloom-filters-explained/
https://www.geeksforgeeks.org/bloom-filters-introduction-and-python-implementation/
https://dergipark.org.tr/en/download/article-file/
https://gsd.di.uminho.pt/members/cbm/ps/dbloom.pdf
https://dridk.me/bloom-filter.html
https://blog.octo.com/le-filtre-de-bloom
https://fr.wikipedia.org/wiki/SHA-256
https://fr.wikipedia.org/wiki/SMD5

Data Extraction

https://www.w3schools.com/tags/tagtitle.asp
https://www.view-page-source.com/
https://stackoverflow.com/questions/22461463/how-can-i-tell-if-the-page-source-is-xhtml-html-or-xml
https://html.com/tags/p/: :text=The-end-of-the-paragraph,a
https://developer.mozilla.org/fr/docs/Web/HTML/Element/p

Trie

https://www.quora.com/What-are-the-advantages-of-Trie-over-TST-Ternary-Search-Tree-and-vice-versa*
https://www.geeksforgeeks.org/trie-insert-and-search/