Projet 1 - Recherche Textuelle

Description du projet

Ce projet consiste à créer deux programmes de recherche textuelle, le premier programme sera naïf et le second sera de type Boyer-Mooyer. Ce projet permettra d'explorer deux approches différentes de la recherche textuelle, mettant en évidence les avantages et les inconvénients de chacun des programmes. Cela mettra en évidence l'importance de l'optimisation quand un très grand nombre de donnée son à analyser. Les utilisateurs pourront saisir des mots-clés ou des phrases, les programmes trouverons ces mots-clés ou phrases et les programmes renverrons la position/le paragraphe qui correspond. A ces programmes un sujet d'application sur le code pénale sera aussi rendu. Le projet inclura également un rapport détaillant le développement du programme, la gestion du projet ainsi que des explication approfondie sur le sujet d'application.

TextSearch

Description du module

Le document TextSearch sert de module pour l'application dans le document recherche_code
Le fichier est séparé en deux parties :

- L'algorithme naif : cette algorithme consiste à comparer un à un les caractère d'un texte avec ceux du motif cherchée. Afin de renvoyé les paragraphes ou est présent le motif.
- L'algorithme Boyer-Moore : cette algorithme s'appuie sur 2 règles : le bon suffixe ainsi que le mauvaise caractère. Cette algorithme permet un parcours plus efficace du texte et offre un gain de temps considérable pour les textes les plus longs

SUJET D'APPLICATION

Utilité du sujet

La recherche textuelle dans le Code pénal français est utile pour diverses raisons : elle permet une compréhension rapide des lois et des peines en vigueur grâce à des recherches rapides et précises, offrant des référence aux professionnels du droit ou pour étayer leurs arguments et leurs décisions. Dans le cadre des entreprise cela peut aussi être utile pour aider à respecter les lois en vigueurs ainsi que la facilité de ne pas devoir apprendre ou lire le code en entier. Elle facilite une analyse approfondie des articles, servant de ressource éducative pour les étudiants en droit et les acteurs judiciaires. De plus, elle peut être mise à jour simplement, permettant de suivre les modifications et ajustements du Code pour s'adapter aux évolutions de la société, garantissant ainsi une interprétation précise et actuelle des lois pénales en France.

SUJET D'APPLICATION

Sources du sujet

Notre sujet d'application vise à parcourir le Code Pénal, pour renvoyer les paragraphes où le motif que l'utilisateur cherche est présent. On aura donc besoin de ces trois documents :

- "TextSearch.py" : le module qui contient toute les fonctions nécessaires à la recherche textuelle, naïve et Boyer-Moore.
- "Code_penal.pdf" : le document officiel du Code pénal français en pdf, nous l'avons récupéré par ce lien : https://codes.droit.org/PDF/Code%20p%C3%A9nal.pdf
- "recherche_code.py" : le document python qui utilise les modules "TextSearch.py" , PdfReader et tkinter. Il permet à l'utilisateur d'entrer sa recherche dans une interface customisée qui renvoie les 15 premiers paragraphes où le motif cherché est présent

SUJET D'APPLICATION

Fonctionnement du sujet

Le document "recherche_code.py" est divisée en quatre étapes :

- Tout d'abord nous importons les modules "TextSearch.py" , "PdfReader" et "tkinter".
- Grâce à la fonction "PyPDF2" nous convertissons le document "Code_penal.pdf" en str python auquel on retire les 9 premières pages de sommaire obsolètes.
- Nous implémentons un tableau python des éléments str indésirables, ce tableau est nécessaire au bon fonctionnement des fonctions de recherche textuelle.
- Nous implémentons ensuite une interface customisée interactive permettant une meilleur compréhension et utilisation des fonctions de recherche textuelle.

BENCHMARK

Première partie avec un long texte

Nous avons réalisé un benchmark pour comparer le temps d'éxécution entre les deux algorithmes (naïf et Boyer-Moore).

En premier lieu, nous avons regardé avec un texte très long (349 pages) et pour plusieurs motifs:

- Le motif "amendes" est un motif test avec 4 occurences dans le texte, on remarque que pour ce motif, le naïf est environ 1.5 fois plus lent soit environ 0.55 secondes.
- Le motif "est" est un motif petit et très présent, pour ce motif on a l'algorithme naïf à nouveau plus lent d'environ 0.5 secondes, soit environ 1.07 fois plus lent.
- Le motif "n'existe pas dans texte'", est un motif n'existant pas dans le texte. Dans ce cas de figure on voit que Boyer-Moore a de nouveau une durée de 0.6 secondes de moins soit 4 fois plus rapide.

BENCHMARK

Première partie avec un long texte

```
>>> benchmarkbm("amendes",texte,globalindesirable)
1.2973558043499906

>>> benchmarknaif("amendes",texte,globalindesirable)
1.8534807557500244

>>> benchmarkbm("est",texte,globalindesirable)
7.5577063955001

>>> benchmarknaif("est",texte,globalindesirable)
8.069524472150079

>>> benchmarkbm("n'existe pas dans texte'",texte,globalindesirable)
0.2199344659999042

>>> benchmarknaif("n'existe pas dans texte'",texte,globalindesirable)
0.8182762575000652
```

BENCHMARK

Deuxième partie avec un court texte

Deuxièmement, nous avons regardé avec un texte court (alphabet avec séparé en deux paragraphes) et pour plusieurs motifs:

- Le motif "abc" est un motif court on remarque que pour ce motif, le naïf est plus rapide d'environ 2 x 10 E-5 secondes, on peut négliger ce résultat.
- Le motif "fghijklmn" est un motif faisant le tiers de la longueur du texte, on remarque ainsi une différence de 6 x 10 E-5 secondes .
- Le motif "n'existe pas dans texte'", est un motif n'existant pas dans le texte et de grande taille par rapport au texte. Dans ce cas de figure on voit que l'algorithme naïf est 10 fois plus rapide que le Boyer-Moore mais dans un ordre de grandeur entre 10 E-4 et 10 E-5 secondes.

BENCHMARK

Deuxième partie avec un court texte

```
>>> benchmarknaif("n'existe pas dans texte'","abcdefghijklmn.\nopqrstuvwxyz",[])
3.801909988396801e-05

>>> benchmarkbm("n'existe pas dans texte'","abcdefghijklmn.\nopqrstuvwxyz",[])
0.0003544806502759457

>>> benchmarknaif("abc","abcdefghijklmn.\nopqrstuvwxyz",[])
5.180709986234433e-05

>>> benchmarkbm("abc","abcdefghijklmn.\nopqrstuvwxyz",[])
3.6060900220036275e-05

>>> benchmarknaif("fghijklmn","abcdefghijklmn.\nopqrstuvwxyz",[])
2.9279950013005873e-05

>>> benchmarkbm("fghijklmn","abcdefghijklmn.\nopqrstuvwxyz",[])
8.546660001229611e-05
```

BENCHMARK

Conclusion

Ainsi, quand la taille du motif se rapproche de celui du texte (partie 2), on observe que l'algorithme naïf est beaucoup plus efficace que l'algorithme de Boyer-Moore. Mais, au contraire quand la taille du texte est très grande par rapport à celle du motif (partie 1) , on observe l'efficacité de l'algorithme de Boyer-Moore comparé à celle du naïf.

TESTS

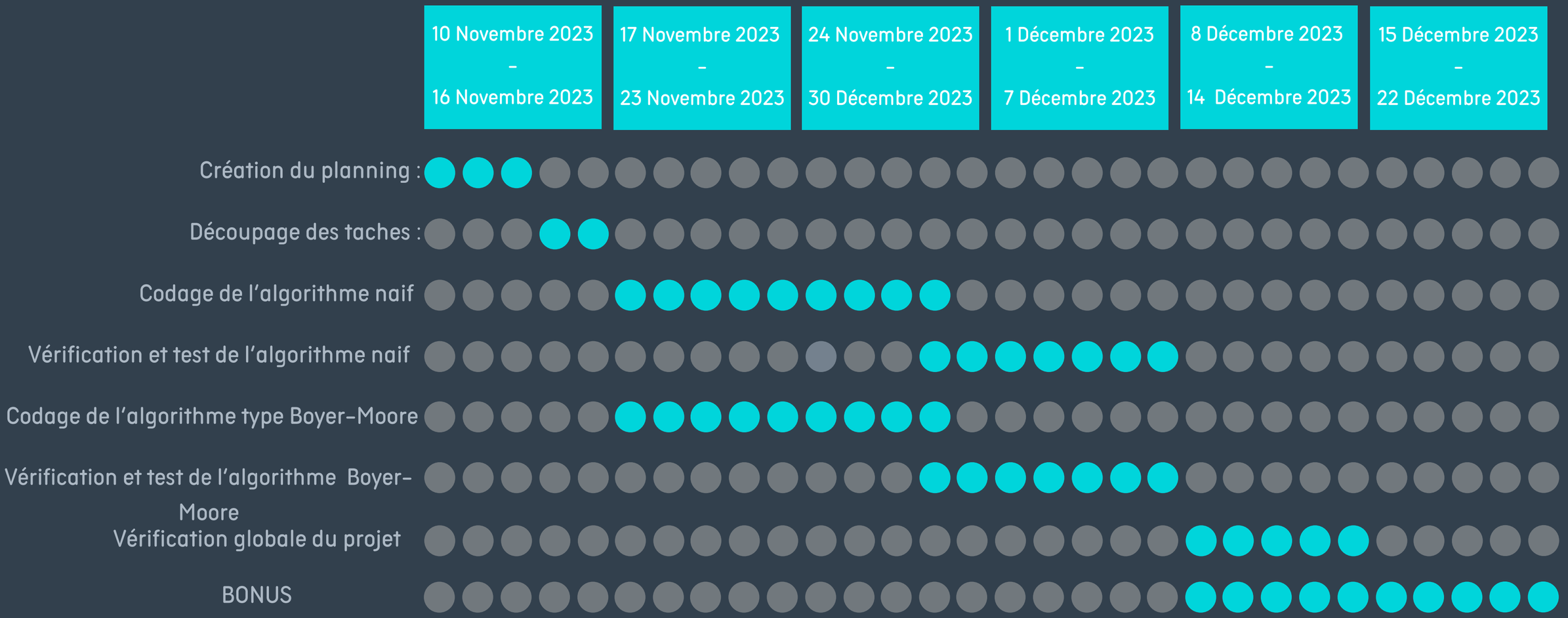
Pour les tests du module TextSearch, nous avons fait un fichier important le module, pour ensuite faire les tests. Cela évite la pollution du module. Les résultats attendus sont marqués dans le programme et l'affichage se réalise à partir de la fonction print.

MODE D'EMPLOI

Tout d'abord, il faut installer le module PyPDF2 dans la console de Python, la manière de le faire dépend de votre système d'exploitation ou encore du programme utilisé (Pyzo, Python, Geany etc). Si après l'installation la console indique que le module n'est pas installée, essayez de redémarrer une nouvelle console. Pour le fichier PDF du code pénal veuillez le téléchargez en cliquant sur le lien présent tout en bas du fichier python recherche_code. Ensuite enregistrez le dans le même répertoire que les programmes python et renommez le Code_penal.pdf.

Enfin, pour recherchez le motif souhaitez entrez le dans la barre de recherche de l'interface graphique qui apparaît après l'execution de recherche_code puis cliquez sur un des deux boutons selon l'algorithme souhaité.

Diagramme de Gantt



Work Breakdown Structure

Répartition des tâches

Rapport

Création du Planning	Alexandre
Création du Work Breakdown Structure	Thomas
Création du partage des tâches	Mahmoud/Vincent

Codage

Codage de l'algorithme naif	Mahmoud/Vincent
Vérification et test de l'algorithme naif	Alexandre
Codage de l'algorithme type Boyer-Moore	Alexandre/Thomas
Vérification et test de l'algorithme de type Boyer-Moore	Vincent

Projet 2 - Recherche Textuelle

ADAMO Thomas || HAMMOUMRAOUI Alexandre || BACCONNIER Vincent || GHRIBI Mahmoud

2023

SOURCES

Rapport	TextSearch	recherche_code
HAMMADACHE Mathieu	https://mathinfo.ovh/Terminale_NSI/E05_Recherche_Textuelle/index.html	https://pypi.org/project/PyPDF2/
	https://fr.wikipedia.org/wiki/Algorithme_de_Boyer-Moore	https://youtu.be/N4M4W7JPOL4?si=wypfoQTT7oYJ5uPi