LES TROPHÉES NSI

Édition 2022

DOSSIER DE CANDIDATURE
PRÉSENTATION DU PROJET

Ce document est l'un des livrables à fournir lors du dépôt de votre projet : 4 pages maximum (hors documentation).

Pour accéder à la liste complète des éléments à fournir, consultez la page Préparer votre participation.

Vous avez des questions sur le concours ? Vous souhaitez des informations complémentaires pour déposer un projet ? Contactez-nous à info@trophees-nsi.fr.

NOM DU PROJET : BeforeTIOBE, de la recherche textuelle à la popularité des langages

> PRÉSENTATION GÉNÉRALE :

TIOBE.com est un index qui permet depuis 2001, de connaitre la popularité des langages de programmation. Mais, avant cette date, peu de données sont disponibles. En se basant sur les textes des magazines « Computer World » et « Byte Magazine » parus de 1967 à 2005 pour le premier et de 1975 à 1998 pour le second, qui constituent une mémoire de l'histoire de l'informatique, nous allons **remonter le temps, là où l'index TIOBE s'arrête**, en utilisant une base textuelle homogène et couvrant une très grande période.

À la suite du chapitre sur les recherches textuelles réalisé en Terminale NSI, nous avons découvert l'algorithme de recherche Boyer-Moore-Horspool et l'avons utilisé pour connaître l'évolution de la popularité de chaque langage depuis 1967. Nous posons comme hypothèse que plus un langage est cité, plus il est populaire.

> ORGANISATION DU TRAVAIL :

Les membres du groupe :

- Hugo Boudry : s'occupe du son et de l'image
- Jasmine Joly : montage vidéo
- Antoine Roussel : responsable du programme et de la documentation

Chacun a réfléchi de son côté pour créer le programme le plus optimisé, celui d'Antoine a été choisi, étant le plus court et le moins long à s'exécuter. Nous avons ensuite retravaillé son algorithme pour l'améliorer. L'objectif était de créer une page html et de regrouper les graphiques et tout obtenir sur un même document. Nous avons ensuite tous écrit le scénario de la vidéo, Hugo s'est occupé des effets sonores et de la réalisation de l'image, Jasmine du montage vidéo et nous avons tous filmé et joué dans la vidéo,

Nous avons travaillé au lycée et en-dehors. La partie programmation sur le logiciel Pyzo et l'écriture du script en groupe avec Google Docs a été réalisé en classe. En revanche le tournage de la vidéo et son montage ont été réalisés en-dehors.

LES ÉTAPES DU PROJET :

- Travail en classe sur le chapitre de la recherche textuelle
- Découverte de l'algorithme Boyer-Moore-Horspool
- Recherche des popularités de langages dans les magazines Computer World et Byte Magazine

- Élaboration de programmes pour créer un graphique représentant la popularité des langages depuis 1967, celui d'Antoine est choisi
- Écriture du script, recherche des effets sonores et visuels pour la vidéo et l'image
- Tournage et montage de la vidéo
- Documentation de notre projet

> FONCTIONNEMENT ET OPÉRATIONNALITÉ :

Nous avons commencé par collecter les textes bruts des revues depuis le site archive.org. Nous avons ensuite concaténé les textes, dans un fichier séparé pour chaque année pour les deux magazines. L'ensemble de la base de données de texte brut pèse environ 1 Go pour Computer World et 370 Mo pour Byte Magazine.

Nous ne pouvions pas les transférer dans votre formulaire, donc voici la base utilisée avec le programme dans un zip pour Computer World. Il vous suffit d'éditer le script Python avec le bon chemin, dans lequel se trouve le fichier dézippé.

https://app.box.com/s/g3cvmgidpb7dzx58r8tu2plmuu9fgwr4

Voici la base de données pour Byte Magazine, il vous suffit de supprimer les fichiers txt par date de Computer World, par ceux de Byte Magazine, en modifiant la dernière ligne du programme avec les années, car le nombre d'année est plus restreint.

https://app.box.com/s/q4nt9ydn00vksrdy16txb9iog3kcmeel

En utilisant l'algorithme de Boyer-Moore, nous décomptons ensuite les occurrences de chaque langage pour chaque année, pour ensuite les représenter dans un graphique pour afficher visuellement les parts de popularité pour chaque langage.

A chaque étape, nous utilisons un point de contrôle qui permet de voir notre avancement et de voir où est notre erreur car dans un si long programme cela peut prendre du temps avant de comprendre ses erreurs. Par exemple nous avons eu le problème avec le langage Ada qui en 1980 faisait planter le programme, à cause de caractère spéciaux.

Pour faire fonctionner le programme, il est nécessaire d'avoir installé le paquet matplotlib sur votre distribution Python, avec pip install matplotlib.

> OUVERTURE :

L'algorithme est assez long à s'exécuter, ce qui est normal, puisque nous analysons plus d'1 Go de texte brut, ce qui est colossal. Nous aurions pu utiliser du calcul parallèle sur chaque cœur pour accélérer le traitement, ou trouver un algorithme plus efficace que Boyer-Moore pour chercher plus vite.

Il serait possible de séparer nos fonctions en différents fichiers et de les appeler comme des modules pour obtenir un programme moins conséquent et plus organisé. De plus utiliser la méthode try-except pourrait nous permettre de contourner les erreurs du programme, notamment pour Ada.

Nous pouvons poster la vidéo tournée et publier nos recherches et nos programmes sur des forums. Ce programme et l'algorithme de Boyer-Moore-Horspool plus précisément peut avoir de nombreuses autres utilités de recherche textuelle et ne sert pas que pour analyser la popularité des langages.

Il aurait peut-être été plus pratique de travailler sur un seul programme tous ensemble et de réunir les meilleures idées en une seule fois plutôt que de travailler chacun de son côté avant de réunir nos recherches car nos programmes n'étaient pas assez optimisés. Chaque programme avait ses avantages et inconvénients et travailler en groupe directement aurait pu nous permettre de réduire le nombre de problèmes.

DOCUMENTATION

Dans ce programme nous utilisons l'algorithme Boyer-Moore. Il consiste à rechercher un motif dans une chaine de caractères. On compare d'abord la dernière lettre du motif avec la lettre au même indice de la chaine de caractères. Si ces deux lettres sont les mêmes, on continue la comparaison du motif avec la chaine de droite à gauche jusqu'à avoir trouvé le mot complet ou qu'une lettre ne corresponde plus. Si les deux lettres ne correspondent pas en revanche, on regarde si la lettre comparée dans la chaine est présente dans le motif cherché. Si c'est le cas, imaginons que le mot mesure n lettres et que la lettre se trouve à l'indice p du motif, alors on décalera le motif le long de la chaine de n-p-1 lettres vers la droite afin que la lettre de la chaine

corresponde à sa place dans le motif. Ensuite on compare la dernière lettre du motif avec la lettre qui lui correspond dans la chaine. En revanche si la lettre comparée n'est pas présente dans le mot, on décale le motif de n lettres vers la droite. On répète ces étapes jusqu'à trouver le mot ou alors que le bout de la chaine soit atteint. C'est un algorithme qui évite les nombreuses répétitions d'un algorithme naïf qui décale seulement d'une lettre à chaque fois le motif.

Afin de rendre plus beau et plus facile notre programme nous avons décidé de le faire en POO (programmation orienté objet). Le but de la POO consiste à définir et faire interagir entre eux des objets, compris ici comme tous types de structures issues d'un langage donné. Ainsi nous avons 3 classes.

```
class Boyer_Moore :
    def __init__ (self,Motif,Chaine):
        self.Motif=Motif
        self.Chaine=Chaine

    def PreTraitement(self):
        dictionnaire = {}
        for i in range(len(self.Motif)-1):
            dictionnaire[self.Motif[i]] = len(self.Motif) - i - 1
        return dictionnaire

    def Moore(self):
            TablePreTraitement = self.PreTraitement()
            ListeRetournee=[]
            IndiceChaine = 0

            while IndiceChaine <= len(self.Chaine)-len(self.Motif):
                IndiceMotif = len(self.Motif)-1


                while self.Chaine[IndiceChaine+IndiceMotif] == self.Motif[IndiceMotif]:
                    if IndiceMotif == 0:
                        ListeRetournee.append(IndiceChaine)
                    IndiceMotif += -1


                if self.Chaine[IndiceChaine+len(self.Motif)-1] not in TablePreTraitement:
                    decalage = len(self.Motif)
                else:
                    decalage = TablePreTraitement[self.Chaine[IndiceChaine+len(self.Motif)-1]]
                IndiceChaine += decalage
            return ListeRetournee
```

En premier la classe Boyer-Moore qui permet de trouver tous les indices d'un mot qu'on lui donne et qui les met dans une liste. Elle contient deux méthodes, une première qui crée la table de prétraitement qui est un dictionnaire contenant les lettres du Motif et les indices de décalages. En effet à l'inverse de l'approche naïve qui compare lettre par lettre la méthode de Boyer-Moore va décaler en fonction de l'emplacement de la lettre dans le mots

Exemple de tables :

Motif	Résultat de `PreTraitement`
« mot »	{'m': 2, 'o': 1}
« yoda »	{'y': 3, 'o': 2, 'd': 1}
« chimie »	{'c': 5, 'h': 4, 'i': 1, 'm': 2}
« physique »	{'p': 7, 'h': 6, 'y': 5, 's': 4, 'i': 3, 'q': 2, 'u': 1}

Enfin la méthode Moore permet de permet de trouver tous les indices d'un mot qu'on lui donne et les met dans une liste en utilisant la table de prétraitement

Passons maintenant à la deuxième classe qui est Classement

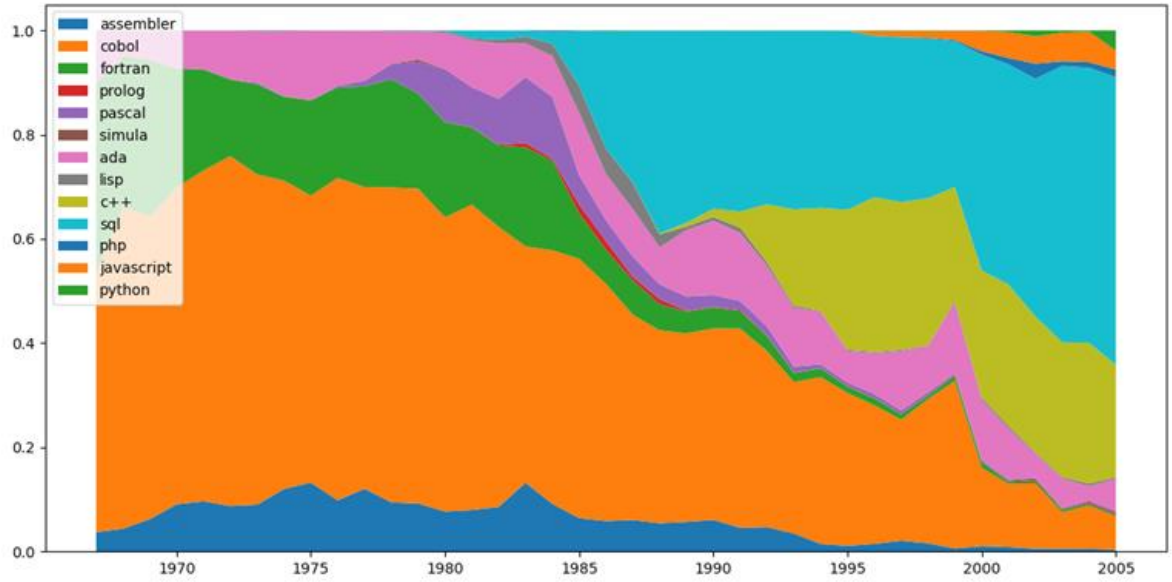
```
class Classement :
    def __init__ (self,ListeDeMotsAchercher,depart,fin,cheminfichier):
        self.ListeDeMotsAchercher=ListeDeMotsAchercher
        self.depart=depart
        self.fin=fin
        self.cheminfichier=cheminfichier
        self.ListeF=[]
        self.x=[]
        self.y=[]
    def ListePourcentage(self) :
        for i in range(self.depart,self.fin+1):
            self.x.append(i)
            Historique=[]
            fichier =open(self.cheminfichier+"ComputerWorld\\" + str(i) + ".txt","r",encoding="utf8")
            Texte=fichier.read()
            for j in self.ListeDeMotsAchercher:
                print(str(j),i)
                a=Boyer_Moore(str(j),Texte)
                Historique.append(len(a.Moore()))
            self.ListeF.append(Historique)
        for k in range (len(self.ListeF)):
            somme=sum(self.ListeF[k])
            for l in range (len(self.ListeDeMotsAchercher)):
                self.ListeF[k][l]=(self.ListeF[k][l]/somme)
        for g in range (len(self.ListeDeMotsAchercher)):
            self.y.append([])
            for h in range(len(self.ListeF)):
                self.y[g].append(self.ListeF[h][g])
        plt.stackplot(self.x,self.y)
        legend = plt.legend(self.ListeDeMotsAchercher)
        legend.set_title("Legende", prop = {'size':1})
        plt.title("Classement des languages de programations")
        cheminimages=self.cheminfichier+"DOCUMENTATION\\ClassmentComputerWorld.png"
        images=plt.savefig(self.cheminfichier+"DOCUMENTATION\\ClassmentComputerWorld.png")
        return cheminimages
```

Cette classe a pour méthode ListePourcentage qui va permettre d'enregistrer une image du graphique qui montre le classement des mots que l'on lui a donné en fonction de leur occurrence dans un magazine ici nous avons fait un classement des langages de programmations en fonctions de leurs occurrence dans un des plus gros anciens magazines d'informatique du monde (COMPUTER WORLD) sur une période choisie. Ainsi en premier la méthode va faire une liste avec le nombre d'occurrences du langages grâce à la liste renvoyer par la class Boyer-Moore et cela pour toutes les dates et tous les langages puis une fois qu'il a tout ça nous allons transformer toutes ces occurrences obtenues en pourcentages afin que ce soit plus lisible sur le graphique. Enfin nous créons le graphique sous la forme de stackplot pour que ce soit plus lisible, compréhensible et joli pour celui qui regarde le graphique puis on enregistre l'image afin de pouvoir faire la suite.

Pour finir nous ne voulions pas que ce soit matplotlib qui affiche l'image car nous ne trouvons pas sa si agréable pour celui qui fait fonctionner le programme ainsi nous créons directement à partir de python un fichier HTML où se trouve l'image. Grâce à la fonction suivante

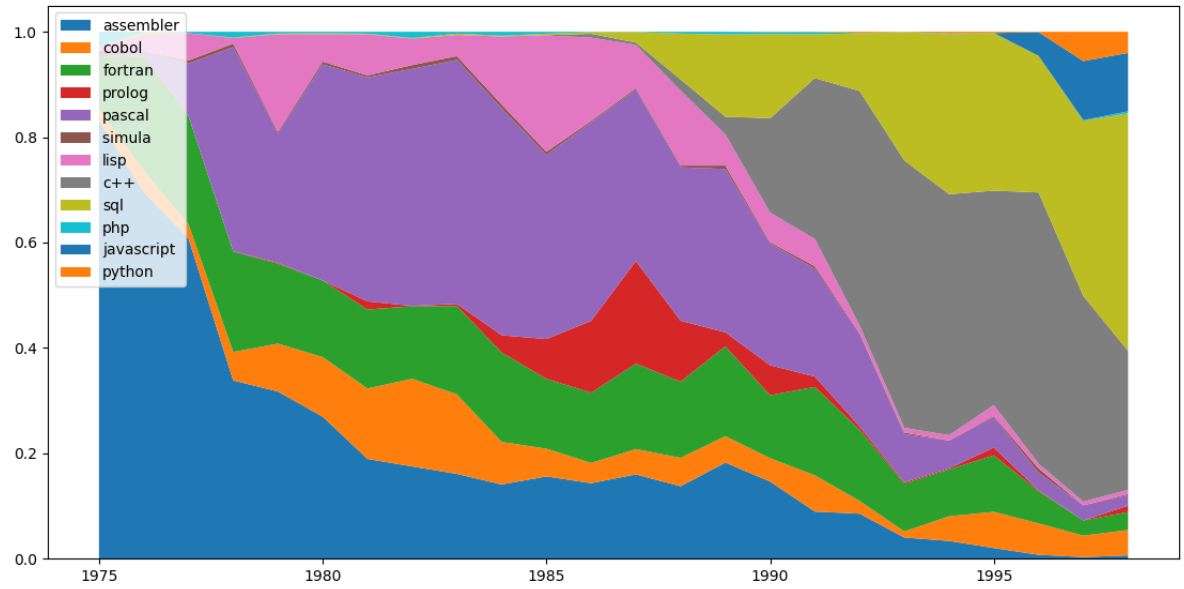
```
def site (classement,depart,fin,cheminfichier):
    h=open(cheminfichier+"DOCUMENTATION\\Classment Computer World.html", "w")
    ClassementDesLanguages=Classement(classement,depart,fin,cheminfichier)
    images=ClassementDesLanguages.ListePourcentage()
    html="<center><img src = " + '"'+ images + '"' +", width = 700"+ ", height=500" + ", loading = 'lazy'></center> "
    h.write(html)
    h.close()
    return "regarder ici pour voir le resultat : E:\\Classment Computer World.html"
```

Les résultats avec Computer World 1967 à 2005 :



On peut définir différentes périodes clés dans l'évolution de la popularité des langages. Tout d'abord, avant 1970, les langages COBOL et FORTRAN sont les plus utilisés mais cèdent leur place dans les années 1980 au langage de base de données SQL, et au début des années 1990, au langage c++. Au début des années 2000, l'éexplosion des langages est perceptible avec javascript et PHP. En ce qui conerne Python, créé en 1991, sa popularité ne devient perceptible qu'à partir de 2004-2005.

Les résultats avec Byte Magazine 1975 à 1998 :



On observe les mêmes tendances de fond, mais dans des proportions différentes, notamment pour COBOL. On peut poser comme hypothèse que l'équipe de rédacteurs étant différente dans les deux magazines Computer World et Byte Magazine, les compétences au sein des rédactions sur chaque langage sont différentes, ainsi que la ligne éditoriale.

Des historiens dans le domaine des sciences numériques pourraient certainement proposer une analyse plus éclairante de ces données.