UNIVERSITÉ DE SHERBROOKE
Faculté de génie
Département de génie électrique et génie informatique

RAPPORT APP5I
MARKOV

Structure de données et Complexité
APP5 informatique

Présenté à
Domingo Palao Muñoz

Présenté par
Pascal-Emmanuel Lachance – lacp3102
Alisée Perrault – pera3307

Sherbrooke – 16 mars 2021

Table of Contents

1. DÉVELOPPEMENT

1.1 DESCRIPTION DES DIFFÉRENTES STRUCTURES DE DONNÉES

La structure de donnée qui a été utilisée pour stocker les mots et leurs fréquences est le tableau de hachage. Dans Python, cette structure a été implémentée à l'aide de dictionnaire. Elle nous permet d'avoir des clés uniques, qui correspondent aux mots et d'avoir accès à la fréquence de ceux-ci. Les dictionnaires ne permettent pas les doublons de clé, ce qui est parfait pour l'utilisation qu'on voulait en faire. Les collisions sont gérées avec la méthode d'adressage ouvert, c'est-à-dire que si deux clés mènent au même index, le code parcourt le dictionnaire jusqu'à ce qu'il trouve une place libre. Les calculs pour trouver la proximité de deux textes ont été effectués sous cette structure.

Pour faire la chaîne de Markov nous avons choisi d'utiliser les Graphs comme structure de donnée. Les Graphs dans Python permettent d'avoir un dictionnaire de dictionnaires. La clé du premier dictionnaire permet de stocker le premier mot d'un binôme. La valeur des clés correspond au deuxième dictionnaire qui contient tous les mots qui peuvent suivre le premier mot du binôme. Le deuxième dictionnaire contient donc le deuxième mot du binôme et la fréquence du binôme. Comme on utilise la structure Graph, on crée des liaisons entre les mots des binômes et on ajoute un poids à cette liaison grâce à la fréquence.

Des listes chaînées ont également été utilisées au long du processus. La fonction de triage pour trier le dictionnaire retourne une liste chaînée. Le désavantage des listes chaînées c'est que la recherche d'éléments peut être longue, car il faut traverser la chaîne à partir du début. Heureusement, l'utilisation que nous avons faite de nos listes chaînées ne nécessitait pas de recherche d'élément. Les éléments à l'intérieur de nos listes chaînées étaient des Tuples, ce qui permet de stocker plusieurs items dans une seule variable.

1.2 ÉVALUATION DE LA COMPLEXITÉ DES ALGORITHMES

L'algorithme d'ajout de mots à une complexité de O(nlogn), chaque ligne se fait interpréter une par une, les caractères non-désirés (ponctuations, espaces, retours de lignes, ect.) sont enlevés par l'outil *split* de la librairie *Regex* du Python, qui est O(n) pour chaque caractère qu'elle analyse, mais qui n'est pas prise en compte dans cette analyse de complexité. Chaque mot, maintenant séparé, est ensuite transformé en minuscules s'il fait plus de 2 lettres de long, et stocké dans un dictionnaire. L'ajout des mots dans le dictionnaire se fait généralement en O(1), bien qu'il soit possible qu'il se fasse en O(n) s'il y a collisions à tous les éléments du dictionnaire. Si la clé représentant le mot existe déjà dans le dictionnaire, sa valeur est incrémentée de 1. La dernière étape est d'ordonner le dictionnaire, ce qui est fait avec la fonction *sorted* du Python, à la complexité O(nlogn) en moyenne et en pire cas (bien qu'elle puisse se faire en O(n)), et qui nous donne une liste de tuples représentant chaque mot et leur nombre d'occurrences dans le texte.

L'étape ayant la complexité la plus grande étant donc O(nlogn), l'algorithme au complet est limité à cette complexité.

1.3 NGRAMMES LES PLUS UTILISÉS

	Unigramme	Bigramme
Balzac	les	dans les
Hugo	les	jean valjean
Ségur	que	des ormes
Verne	les	dans les
Voltaire	les	tous les
Zola	les	elle avait

1.4 NOMS DES AUTEURS DES DIFFÉRENTS TEXTES À IDENTIFIER

Nom du texte	Auteur probable (avec unigrammes)	Auteur probable (avec bigrammes)
Texte0.txt	Zola	Zola
Texte1.txt	Ségur	Ségur
Texte2.txt	Hugo	Hugo
Texte3.txt	Voltaire	Voltaire
Texte4.txt	Balzac	Balzac
Texte5.txt	Verne	Verne

Annexe A – RAPPORT GÉNÉRÉ AUTOMATIQUEMENT

Mais qui permet stocker les graphs comme utilise structure donnée qui est d'ordonner dictionnaire permet stocker plusieurs items dans python complexité plus lettres long stocké dans dictionnaire jusqu'à qu'il trouve une liste chaînée. Cette analyse mais qui correspondent aux mots des listes chaînées c'est dire que deux clés uniques qui n'est pas les graphs dans texte dernière étape est d'ordonner dictionnaire valeur des listes chaînées ont été effectués sous cette analyse mais qui nous avons choisi d'utiliser les mots dans dictionnaire fait avec fonction *sorted* Python permettent pas les mots leurs fréquences est fait avec méthode d'adressage ouvert. L'utilisation qu'on voulait faire qui permet stocker plusieurs items dans cette structure donnée qui est limité cette analyse mais qui été effectués sous cette structure donnée qui correspondent aux mots qui n'est pas les mots qui nous donne une complexité nlogn l'algorithme complet est d'ordonner dictionnaire qui permet stocker les mots une seule variable comme utilise structure été effectués sous cette complexité nlogn chaque mot