Département d'Informatique
2ème année de Licence

Compression Lempel-Ziv-Welch

Rapport de projet de Structure de donnnées II

MOHAMED YOUNIS Ahmed & MAHAMAT Ouagal

Décembre 2021

Enseignants : HETROY WHEELER Franck,
HAAS Antoine,
MORGENTHALER Sébastien.

Sommaire

I- Introduction

La compression Lempel-Ziv-Welch (LZW) est une méthode de compression de données non destructive. C’est-à-dire, qu’il n’y aucune perte de données lors de la décompression des données codées(compressées).Cette méthode de compression est une amélioration par Terry Welch de l’algorithme inventé par Abraham Lempel et Jacob Ziv en 1978.

Le principal objectif de la compression LZW est d’éviter les répétitions dans les chaînes de caractères du fichier à compresser, afin d’économiser de la place.L’algorithme utilise une table de traduction, que nous appellerons dictionnaire qui se construit dynamiquement, au cours de la compression mais aussi de la décompression.

II- La gestion du dictionnaire

Les algorithmes de compression et décompression utilisent tous les deux un dictionnaire. Nous avons représenter ce dictionnaire sous trois formes (liste chaînée, Trie informatique, et table de hachage).

Avant de créer le dictionnaire nous récupérons la taille du fichier a compresser et nous allouons le dictionnaire selon cette taille pour éviter le plus possible les erreurs d'insuffisances d'espace pour le codage ou de gaspillage de mémoire inutile.

Et enfin nous avons aussi choisi de ne considérer que les caractères Extended ASCII de 0 à 255 pour la compression et la décompression.

III- Mises en applications et Analyse

Nous vous proposons une mise en application des algorithmes de compression et de décompression dans le fichier main.c. Celle-ci prend en paramètre 3 fichiers (le fichier texte a compressé, un fichier au format .lzw qui contient le code binaire de la compression et un fichier texte qui comprendra le texte décompressé a l'aide de l'algorithme de la décompression)

Nous vous avons aussi fourni un jeu de testes sur l'insertion et la recherche pour toutes les trois structures au vu de déterminer laquelle est la plus efficace en terme de rapidité . Les résultats sont illustrés sur les figures dans l'annexe.

Pour les insertions nous avons obtenu pour la liste un temps de 0.000044 ms pour le trie 0.000009 ms et pour la hashmap 0.000177ms.

Nous en avons déduit que la compression LZW en utilisant comme structure la hashmap est la plus rapide parmi les trois suivi du Trie et en dernier la liste. Tout cette théorie pourra être vérifier lors de l'exécution du main en utilisant les différentes structures.

IV- Programmation modulaire

Afin de se séparer aux mieux nos tâches, nous avons découpé notre projet en différentes parties, appelées "modules". Chaque module est découpé en deux fichiers :

- un fichier contenant les prototypes des différentes fonctions ainsi que les déclarations des structures de données (en .h) dans le dossier Includes,
- un fichier contenant l'algorithme des fonctions (en .c) dans le dossier Src.

Nous avons donc cinq modules :

- liste : ce module contient la déclaration de la structure de donnée sous forme de liste chaînée et les fonctions d'initialisation d'insertion et la recherche dans un dictionnaire implémenté en utilisant la structure liste.
- trie : contient la déclaration de la structure de donnée sous forme d'un trie informatique et les fonctions d'initialisation d'insertion et la recherche dans un dictionnaire implémenté en utilisant la structure Trie.
- hashmap : contient la déclaration de la structure de donnée sous forme d'un tableau associative appelé tables de hashage et les fonctions d'initialisation d'insertion et la recherche dans un dictionnaire implémenté en utilisant la structure hashmap.
- Main : qui contient une mise en application de la compression et la décompression

L'exécution du programme et l'appel des différents modules seront fait à l'aide du fichier main.c. Ce module nous permettra ainsi de lancer la compression et la décompression de nos fichiers.
La compilation se fera à l'aide d'un Makefile fourni, nous compilerons en utilisant la commande make et nous nettoierons en utilisant la commande make clean.

V- Conclusion

À travers ce projet, nous avons pu constater les efforts à fournir ainsi que les nombreuses difficultés qu'il est possible de rencontrer lors de la programmation d'algorithmes. Néanmoins, la répartition des tâches en fonction des différents aspects du programme nous a permis de progresser comme nous le souhaitions.
Nous avons pu nous rendre compte de l'importance de bien comprendre les algorithmes (en particulier la compression et la décompression) afin de fournir un travail conforme aux consignes de départ.
Une conséquente partie du travail consistait à la gestion du dictionnaire, sa façon de le concevoir en terme de structure, de taille, puis en terme de relations avec les deux algorithmes (l'ajout de mot, la recherche,...).

Nous avons donc pu, au terme du projet, confirmer l'efficacité de l'algorithme LZW et donc son intérêt pour la compression d'un texte.

Finalement, nous pensons que cette expérience de travail en binôme a été très enrichissante, du point de vue de l'organisation à avoir, du travail à répartir, de la conception du code au cœur du programme, de nos différentes visions sur la manière de procéder.

VI- Annexes

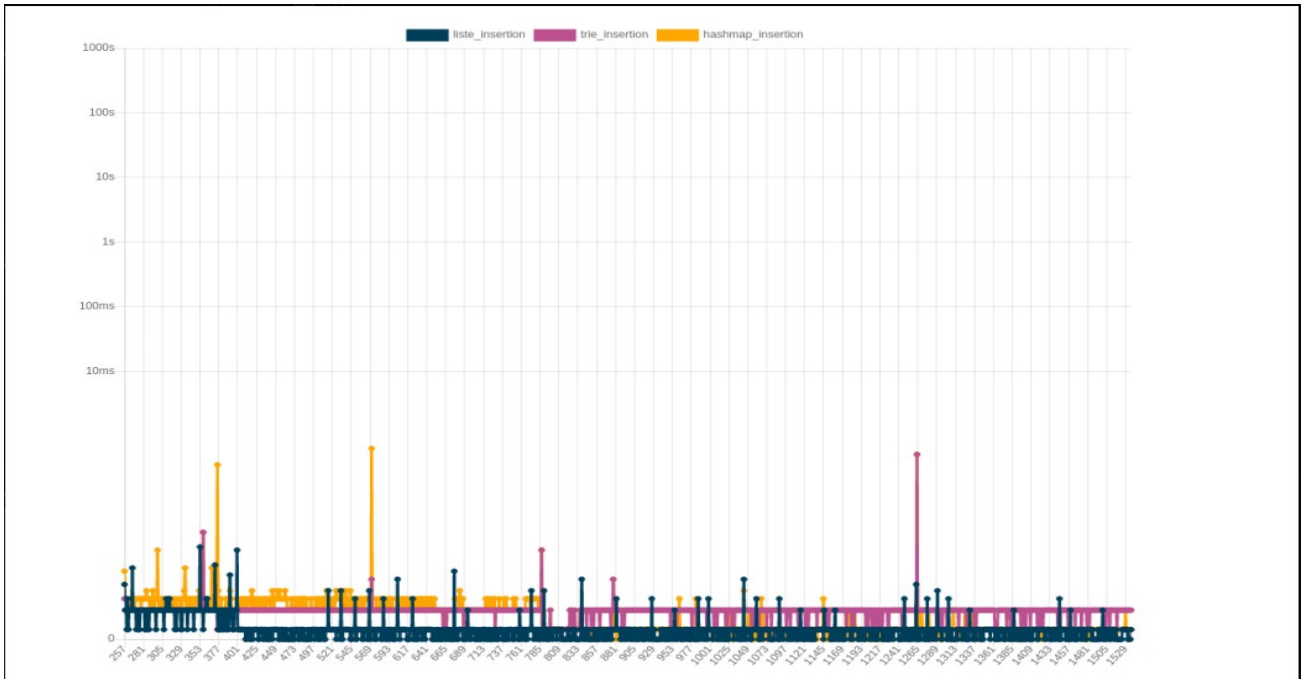

figure 1:les insertions

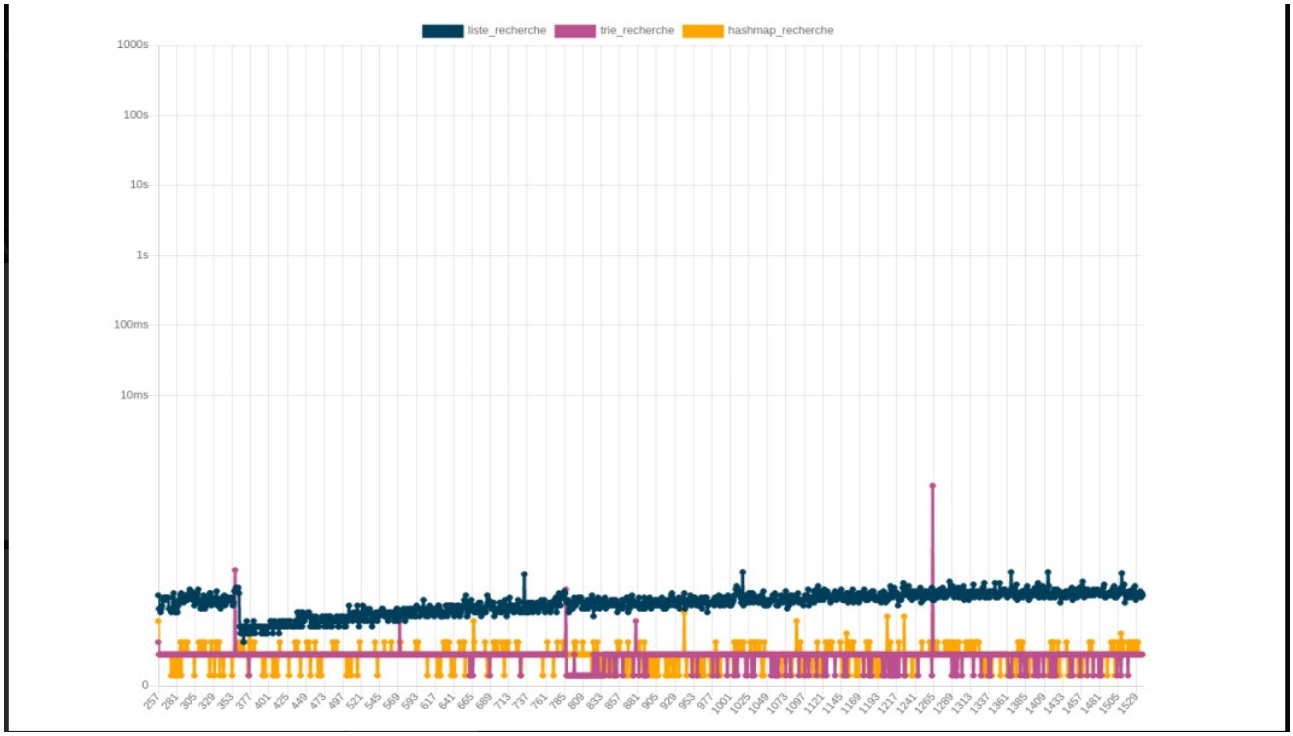

figure 2:les Recherches