université
PARIS-SACLAY

FACULTÉ
DES SCIENCES
D'ORSAY

RAPPORT DE PROJET

Monsieur Kim Nguyễn
Responsable de l'Unité d'Enseignement

Compression de texte

Méthode de Huffman

Alexis VO - Théo G[redacted]

Licence 2 Double Diplôme
Informatique Mathématiques

Novembre 2024 – Janvier 2025

TABLE DES MATIÈRES

INTRODUCTION

Le projet consiste à implémenter un système de compression de fichiers en utilisant l'algorithme de Huffman. Ce projet inclut la gestion des fichiers binaires, la construction d'un arbre de Huffman pour encoder les données et la décompression de ces fichiers compressés. Nous avons également mis en œuvre une fonctionnalité d'affichage des statistiques sur les fichiers traités, telles que le nombre de lettres, de mots, la lettre la plus fréquente, ainsi que le taux de compression obtenu. Une fonctionnalité d'aide est aussi présente.

TYPES DE DONNÉES

1. Arbre de Huffman

L'arbre de Huffman est utilisé pour la compression des données. Il est construit en utilisant une table de fréquences des caractères du fichier d'entrée. Chaque caractère est représenté par un code binaire, où les caractères les plus fréquents ont des codes plus courts. Chaque nœud représente un caractère et sa fréquence.

2. File de priorité

Nous avons utilisé une file de priorité (heap) pour constuire l'arbre de Huffman. Cette file nous a permis de sélectionner efficacement les deux nœuds ayant les fréquences les plus faibles pour les fusionner, et ce, jusqu'à ce qu'un seul nœud reste. La file de priorité est implémentée via un tas binaire, où les éléments sont insérés en maintenant l'ordre de priorité basé sur les fréquences des caractères.

3. Autres types auxiliaires

- **Table de codage :** une liste de paires (caractère, code binaire) utilisée pour encoder et décoder les données.
- **Chaînes binaires :** utilisées pour représenter les octets du fichier sous forme binaire.
- **Octets :** après compression, les données sont regroupées en octets afin de les écrire dans un fichier binaire.

EXEMPLE DE FONCTION COMPLEXE

Pour l'exemple, nous avons choisi de décrire la fonction `decompress_file` qui permet de décompresser un fichier binaire en utilisant une table de codage stockée dans ce fichier.

Description de la fonction

Illustrons cela par un exemple…

Imaginons que nous avons un fichier compressé, example_compressed.txt, et nous voulons le décompresser dans un fichier example_decompressed.txt.

Disons que ce fichier contient les données suivantes :

- Le nombre total de caractères à décompresser : 10 (donc, il y a 10 caractères au total dans le fichier compressé).
- Le nombre de codes dans la table de codage : 3 (donc, il y a 3 codes distincts dans la table de codage).
- La table de codage contient des informations telles que :
    - Le code 0 correspond au caractère a.
    - Le code 1 correspond au caractère b.
    - Le code 2 correspond au caractère c.
    - La séquence compressée dans le fichier est : 0 1 2 1 0 2 0 1 1 0 (ces codes seront utilisés pour reconstruire le texte original).

Étapes de la décompression

1) Ouverture des fichiers

- Le fichier example_compressed.txt est ouvert en lecture binaire, et example_decompressed.txt est ouvert en écriture binaire.

2) Lecture des informations

- Le nombre total de caractères est lu comme étant 10.
- Le nombre de codes dans la table de codage est lu comme étant 3.

3) Lecture de la table de codage

- La table de codage est lue à partir du fichier d'entrée, et elle contient les associations : 0 -> a, 1 -> b, 2 -> c.

4) Décompression des données

- La fonction decode_et_ecrit_large décode les codes en utilisant la table de codage :
    - Le code 0 devient a.
    - Le code 1 devient b.
    - Le code 2 devient c.

Après avoir décompressé toute la séquence, le texte résultant est : "abcbacbba".

5) Fermeture des fichiers et fin

- Le fichier de sortie example_decompressed.txt est fermé, ainsi que le fichier d'entrée example_compressed.txt

- Un message est affiché pour confirmer la fin de la décompression.

JEUX DE TESTS

Pour tester nos programmes, nous avons réalisé différents jeux de tests dont en voici quelques uns :

À l'aide de la commande suivante, on compresse un fichier contenant le mot "satisfaisant" :

./huff.exe --compress ./tests/test_mot_satisfaisant.txt ./tests/test_mot_satisfaisant_com.txt

Ensuite, nous avons remarqué que pour de petites tailles, les fichiers compressés sont plus gros que des fichiers non-compressés. Cela pourrait s'expliquer par l'en-tête introduit des fichiers compressés et plus généralement du modèle du code de Huffman.

Pour décompresser le fichier compressé, on écrit la commande suivante :

./huff.exe --decompress ./tests/test_mot_satisfaisant_com.txt ./tests/test_mot_satisfaisant_dec.txt

Et le fichier décompressé se trouve dans le dossier tests et se nomme test_mot_satisfaisant_dec.txt.

Pour tester notre programme sur de gros fichiers, nous avons téléchargé la Bible et le Coran en version texte. Puis, nous avons essayé de compresser les deux fichiers. Pour la compression, il n'y a pas eu de problèmes majaures. Néanmoins, pour la décompression, cela a pris plus de temps. Parfois, le terminal était obligé de "kill" notre programme. Il nous a donc fallut revoir la logique de certaines fonctions, notamment les fonctions récursives, et surtout, la concaténation avec le symbole ^ qui ne doit pas être répété de trop nombreuses fois sinon le programme est coûteux en temps et mémoire.

Enfin, pour vérifier s'il n'y avait pas de différence entre le fichier d'avant compression et celui après décompression, nous avons utiliser la commande Unix `*diff*`

Nous avons également testé la fonctionnalité --stats qui affiche le nombre de lettres, le nombre de mots, la lettre la plus fréquente et si un deuxième fichier est fourni en argument, le programme affiche le taux de compression.

```
Nombre de lettres : 3440635
Nombre de mots : 822064
Lettre la plus fréquente : e (438084 occurrences)
Taux de compression : 61.85%
```

Exemple avec bible.txt

RÉPARTITION DU TRAVAIL

Le travail a été réparti de manière équitable entre les membres du binôme. Nous avons commencé à coder “côte à côte” sur des séances de quatres heures en utilisant LiveShare qui nous offre la possibilité de co-éditer ou encore partager des termainaux.

Nous avons poursuivi l’avancée du projet grâce à git en vérifiant à chaque fois qu’est-ce qui a été implémenté, comment il l’a été et si des améliorations sont possibles. De fait, chacun a participé a l’ensemble du projet.

CONCLUSION

Ce projet a permis de développer une compréhension approfondie des algorithmes de compression, en particulier l’algorithme de Huffman, et de la gestion des fichiers binaires en OCaml. L’implémentation des statistiques et du taux de compression a ajouté une fonctionnalité utile pour évaluer l’efficacité de la compression. Bien que le projet ait présenté des défis, notamment en termes de gestion de la mémoire et de la lecture des fichiers, il a permis de développer des compétences solides en algorithmique et en programmation.