**P A I X – T R A V A I L - P A T R I E P E A C E – W O R K - F A T H E R L A N D**

**Université de Douala The University of Douala**

**Faculté des Sciences Faculty of Science**



**Projet 437**

**THÈME: IMPLÉMENTATION DES CODAGES DE HUFFMAN POUR LA COMPRESSION DES FICHIERS**

|  |  |  |  |
| --- | --- | --- | --- |
| **N°** | **Noms et prénoms** | **Matricule** | **Téléphone** |
| **1** | **KAMDEM YOUMBISSI Chrisppo** | **18S08357** | **698969992** |

**Sous la direction de : Mr NDJE Jonathan**

**ANNÉE ACADÉMIQUE : 2024/2025**

# **Rapport : Compression de fichiers avec le codage de Huffman**

## **1. Analyse**

L'objectif principal de ce projet est de compresser des fichiers en utilisant le codage de Huffman. Cette méthode, basée sur les algorithmes gloutons, réduit efficacement la taille des données en attribuant des codes binaires plus courts aux caractères fréquents et des codes plus longs aux caractères moins fréquents.  
Les étapes essentielles de la compression incluent :

1. Calcul des fréquences des caractères : Identifier combien de fois chaque caractère apparaît dans le fichier.  
2. Construction de l'arbre de Huffman : Utiliser un tas pour générer un arbre binaire optimal où chaque feuille représente un caractère.  
3. Génération des codes : Parcourir l'arbre pour produire les codes binaires associés à chaque caractère.  
4. Encodage : Convertir les caractères du fichier en une séquence de bits basée sur les codes générés.  
5. Sauvegarde des données compressées : Stocker les données compressées sous forme binaire avec les métadonnées nécessaires pour la décompression.  
  
Ce projet vise également à inclure un processus de décompression pour valider l'intégrité des données compressées.

## **2. Conception**

Le projet repose sur une architecture modulaire pour assurer la clarté et la réutilisabilité du code. Les principaux composants sont :

1. Lecture et analyse du fichier : Lire les données du fichier texte et calculer les fréquences des caractères.  
2. Construction de l'arbre de Huffman : Utiliser une structure de tas pour construire efficacement l'arbre.  
3. Génération des codes binaires : Parcourir récursivement l'arbre pour attribuer un code binaire à chaque caractère.  
4. Compression et sauvegarde : Encoder les données en utilisant les codes générés, ajouter un padding pour aligner les données sur des octets complets et sauvegarder les données compressées et les métadonnées dans un fichier binaire.  
5. Décompression (optionnelle) : Lire les données compressées, reconstruire l'arbre de Huffman à partir des métadonnées et décoder les données binaires pour restituer le fichier original.

## **3. Réalisation (en Python)**

L'implémentation suit les étapes de conception décrites ci-dessus. Voici une vue d'ensemble des modules principaux :

· 3.1 Construction de l'arbre de Huffman : L'implémentation inclut la gestion des fréquences et l'utilisation d'une structure de tas pour optimiser les performances.

· 3.2 Compression : Les données sont encodées et sauvegardées sous forme binaire. Les métadonnées (codes) sont sérialisées pour garantir la possibilité de décompression.

· 3.3 Décompression : La reconstruction de l'arbre et le décodage des données binaires sont réalisés pour valider l'algorithme.

. 3.4 Fonction principale : Fonction principale qui gère les paramètres et execute l’algorithme.

· 3.5 Taux de compression : Le calcul du taux de compression permettra d’évaluer l’éfficacité de l’algorithme.

## **4. Résultats**

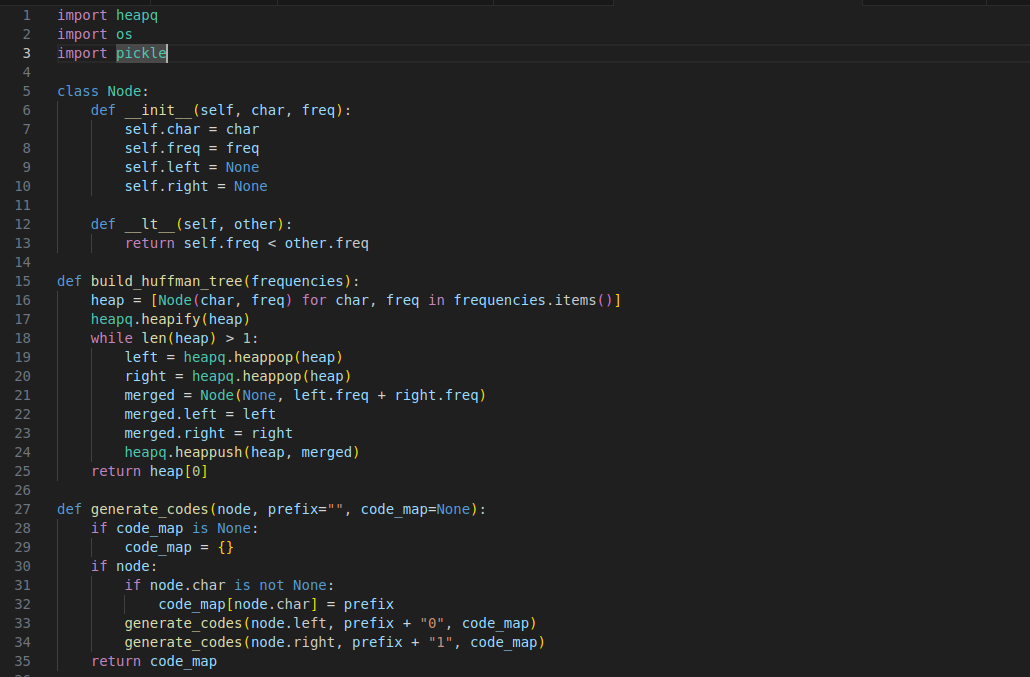
La compression a été testée sur deux fichiers texte. Voici un résumé des résultats :

|  |  |  |  |
| --- | --- | --- | --- |
| Nom du fichier | Taille originale | Taille compressée | Réduction |
| sample.txt | 11 Ko | 8 Ko | ~27% |
| sample\_lourd.txt | 1.6 Mo | 952 Ko | ~41% |

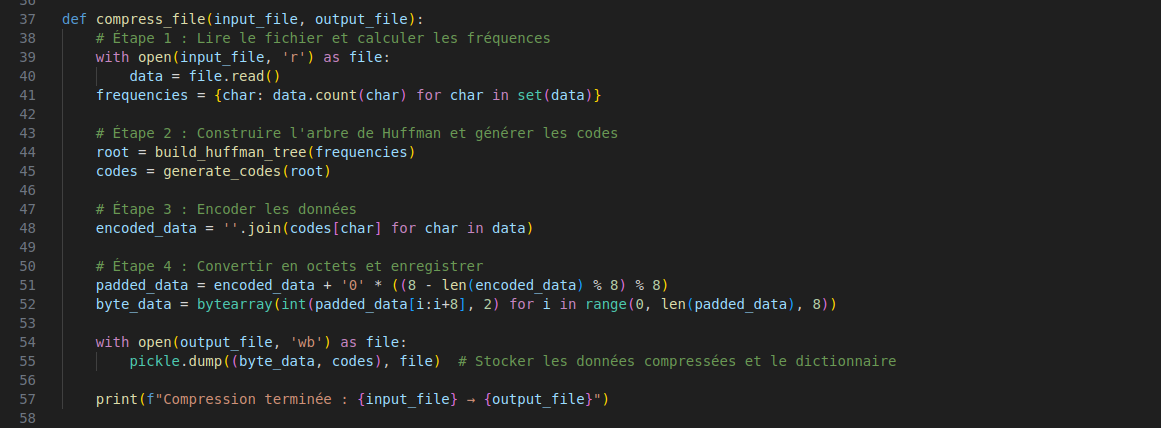
Les résultats montrent que la compression de fichiers texte est efficace.

## **5. Listings de code**

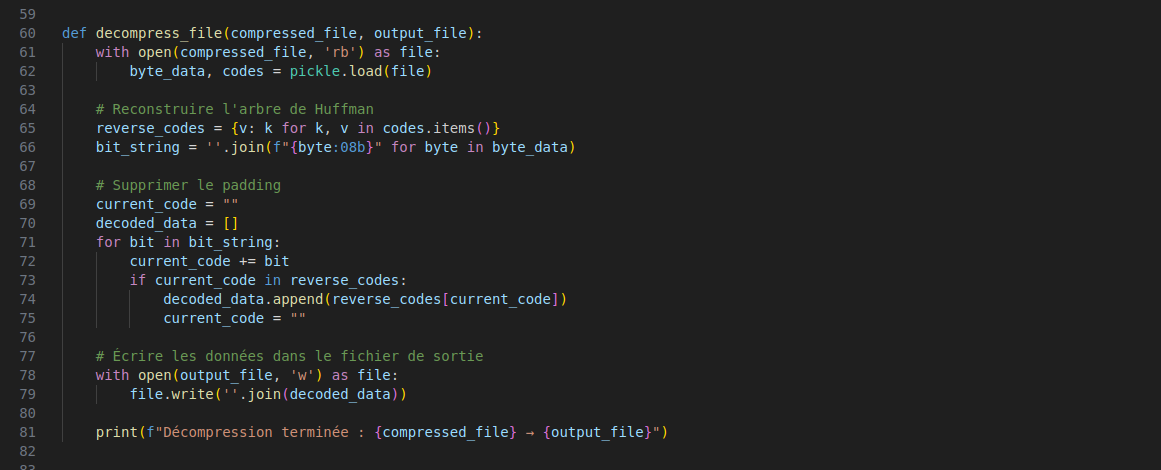
1. Construction de l’arbre



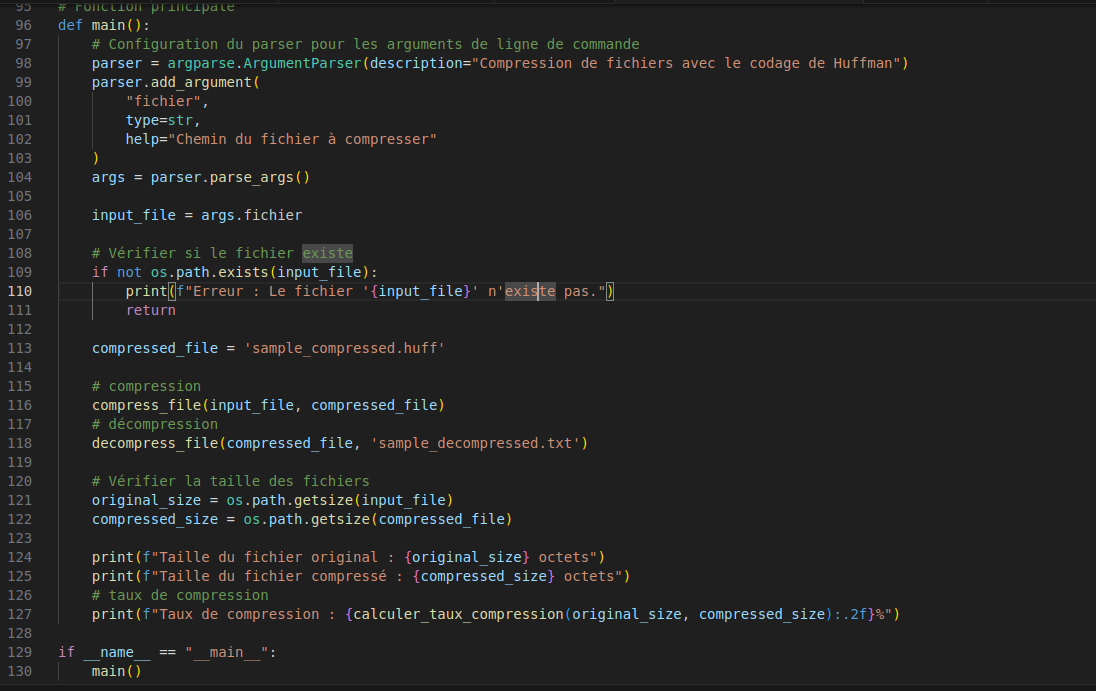
1. Compression



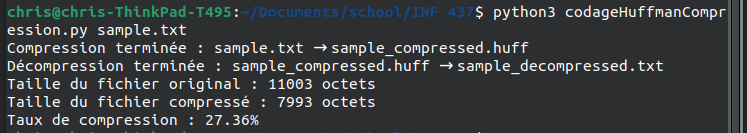
1. Décompresssion



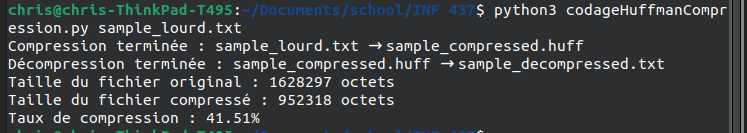
1. Fonction principale



1. Exemple d’utilisation sur fichier leger



1. Exemple d’utilisation sur fichier lourd



## **Conclusion**

Ce projet a permis d'appliquer efficacement le codage de Huffman pour la compression de fichiers, démontrant la puissance des algorithmes gloutons dans l'optimisation des ressources. La solution implémentée est extensible et pourrait être adaptée à d'autres types de données ou à des formats binaires.