L’encodage d’Huffman | Journal de Bord

Lien au projet
- Replit - https://replit.com/join/yhdosmlzbe-smoll24
- https://replit.com/@smoll24/EncodageHuffman?v=1#main.py (Il prend parfois un peu de temps a commencer la première fois que vous exécutez le programme)

La tâche à réaliser :

- Créer un programme en Python pouvant compresser un fichier texte en utilisant la technique d’encodage de Huffman
- Il faudra produire l’arbre ayant servi à l’encodage

- **Conseil :** commencez par de petits fichiers de test (une dizaine de caractères, une phrase) pour vous entraîner avant d'essayer de compresser de gros volumes de texte.

- Il devra pouvoir fonctionner sur un fichier texte de plusieurs milliers de lignes (écrit en anglais ou en français), voire plus.
- Un fichier texte au format UTF-8 correspondant à une œuvre littéraire pourra être téléchargé à partir du site web « Project Gutenberg », une bibliothèque en ligne de livres électroniques gratuits (Top 100 | Project Gutenberg).
- Project rendu (complet ou non), sous la forme d’un fichier .py pour le programme et d’un journal de bord

Journal de bord :

Il indiquera:
- la planification des différentes étapes de la réalisation du projet
- les tâches effectuées par chacun
- les difficultés que vous avez rencontrées
- les essais infructueux et l’aide utilisée
- si le projet est complet ou si des fonctionnalités n’ont pas pu être réalisées

Arbre binaire :

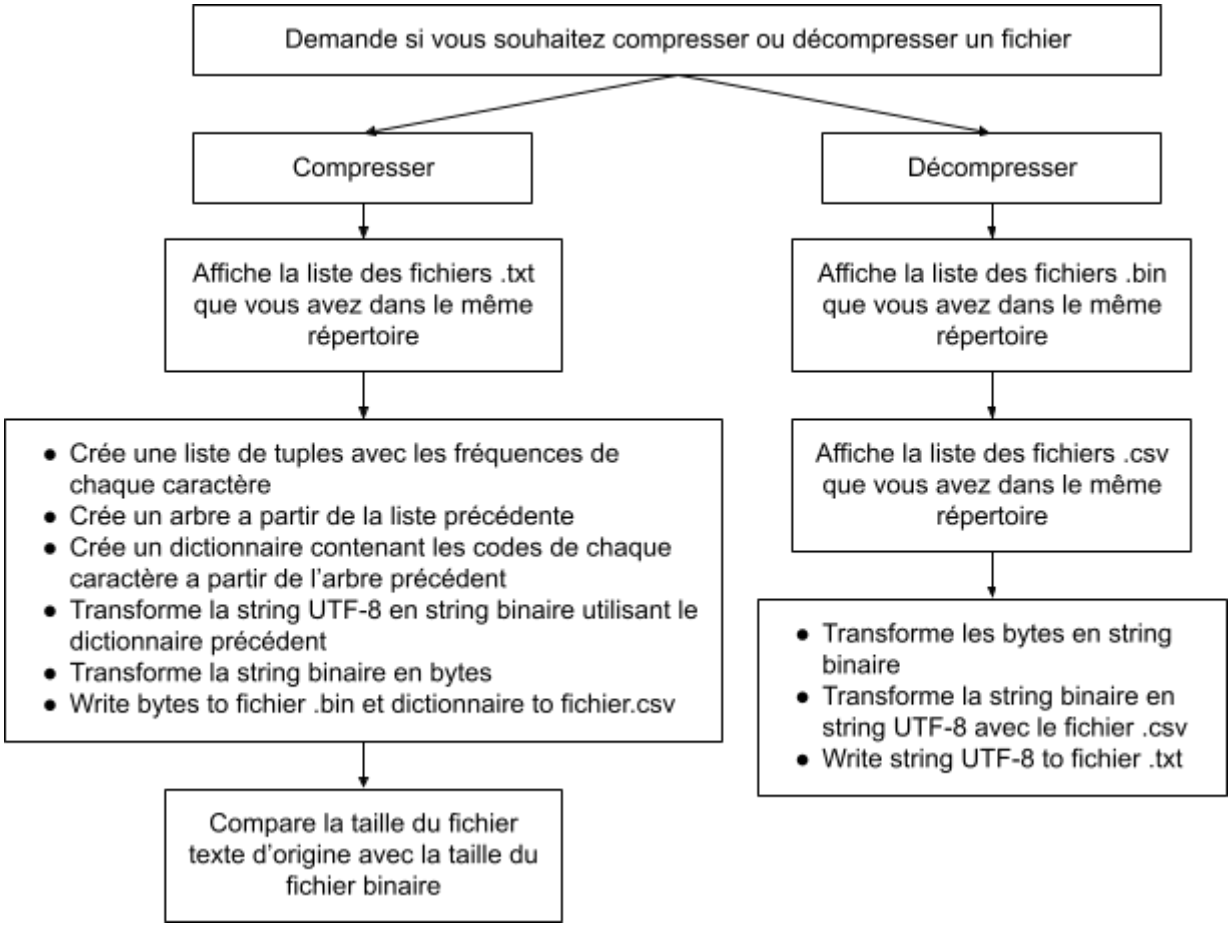
- Affecter un code binaire de longueur variable à chaque caractère en construisant un arbre binaire étape par étape.
- On affecte des valeurs aux arêtes de l’arbre construit selon la règle suivante :
  - l’arête reliant le sous-arbre gauche à la racine est affectée de la valeur 0,
  - l’arête reliant le sous-arbre droit à la racine est affectée de la valeur 1.

Planification des étapes

Etapes

1. Créer une classe arbre. ✅
2. Calculer la fréquence des caractères dans le texte. ✅
3. Créer un arbre avec ces fréquences. ✅
4. Associer un code binaire a chaque caractère correspondant aux valeurs portées par les arêtes du chemin à parcourir depuis la racine jusqu'à la lettre. ✅
5. Encoder chaque caractère dans son code binaire et l'ajouter à un fichier binaire. ✅
6. Lire un fichier .txt et le transformer en fichier .bin avec fichier .csv. ✅
7. Transformer le fichier .bin avec le fichier .csv en nouveau fichier .txt. ✅
8. Vérification de la taille des fichiers. ✅
9. Création d'une interface pour facilement utiliser notre programme. ✅

Plan du Projet



Timeline

Novembre 16, 2023 | 11/16/2023

Etapes

▶️ Créer une classe arbre.
▶️ Calculer la fréquence des caractères dans le texte.
▶️ Créer un arbre avec ces fréquences.

- Création de la structure noeud (Tony)
- Création de la fonction get_frequency(phrase): Trouve la fréquence de chaque caractère et les ajouter à un dictionnaire (Samy)
- Ajoute chaque caractère et sa fréquence dans une liste de tuples ordonnée (Tony)
- Création de la fonction liste_to_arbre(liste): Prend la liste créée et ajoute les deux premiers éléments commes sous-arbres d'une nouvelle instance de la classe Noeud (Samy)
- Création de la fonction somme(liste): Renvoie la somme des deuxièmes éléments des deux premiers tuples dans la liste pour calculer la valeur de la racine du nouveau noeud (la somme des valeurs de ses fils) (Tony)
- Création de la fonction compare(liste): Compare les deuxièmes éléments des deux premières tuples dans la liste pour trouver quel fils a placer a gauche et lequel a droite et renvoyant un tuples avec les valeurs des deux fils du nouveau noeud (Tony)
    - Difficulté: Si on ne garde que les deuxièmes éléments des tuples, les sous-arbres gardés dans le premier élément du tuple est perdu
    - Solution: Ajout de deux ternary expressions pour ajouter le premier élément au tuple et non le deuxième élément s'il s'agit d'une instance de la classe Noeud (Samy)
    - Difficulté: Si on ne renvoie que le premier élément du tuple s'il ne s'agit pas d'une instance de la classe Noeud, on perd la lettre qui correspond à la valeur
    - Solution: Renvoie le tuple en entier et non seulement le premier élément (Samy)

- Fonction liste_to_arbre(liste): Après la création du nœud, enlève les deux premiers éléments de la liste originale. Puis, insère le nœud dans l'arbre pour que la liste reste ordonnée. (Samy)

    - Difficulté: Si on ajoute simplement le nœud dans l'arbre et puis on l'ordonne après avec "sorted", il ajoute l'arbre après les autres éléments avec la même valeur, mais on veut que l'arbre soit placé avant les autres éléments avec la même valeur.
    - Solution: On insère l'arbre directement dans la liste en la traversant avec une boucle while. (Samy)

- Création d'une fonction construction_arbre(phrase) pour appeler la fonction liste_to_arbre(liste) jusqu'il n'y reste qu'un élément dans la liste et puis retourne le nœud racine de l'arbre. (Samy)

Novembre 17, 2023 | 11/17/2023

Etapes

▶️ Associer un code binaire a chaque caractère correspondant aux valeurs portées par les arêtes du chemin à parcourir depuis la racine jusqu'à la lettre.
▶️ Lire un fichier .txt et créer un nouveau fichier .txt.

- Création d'une fonction parcours_liste_lettre(arbre, letter, path=[]) pour trouver le code binaire d'une lettre en parcourant l'arbre, ajoutant des 0 ou 1 a une liste "path" et retournant la liste (Tony)

- Création d'une fonction convert_text_binary(text) pour lire un fichier .txt et le transformer en string (Aide avec Read File as String in Python) et puis créer un nouveau fichier .txt avec une chaine de caractère comportant le codage binaire (Aide avec Print string to text file - python) (Samy)

Novembre 18, 2023 | 11/18/2023

Etapes

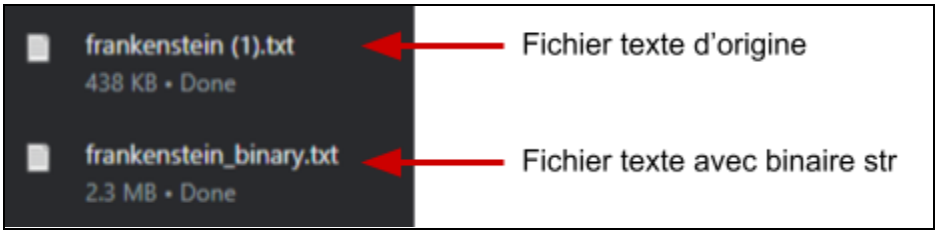
▶️ Encoder chaque caractère dans son code binaire et l'ajouter à un fichier binaire.
▶️ Lire un fichier .txt et le transformer en fichier .bin.
▶️ Transformer le fichier .bin en nouveau fichier .txt.

- Création de la fonction codes_binaires(arbre, liste): Retourne un dictionnaire “codes” contenant les codes en forme de string de chaque caractère présent dans le texte (Aide avec python - How to convert list to string) (Samy)

- Création de la fonction phrase_binaire(phrase, codes): Transforme chaque caractère dans le texte en son codage correspondant en binaire grâce au dictionnaire “codes” et les ajoute a une string contenant l'entièreté du texte en binaire (Samy)

- Création de la fonction phrase_lettres(binaire, codes): Transforme chaque code binaire dans le binaire en son caractère correspondant en UTF-8 grâce au renversement du dictionnaire “codes” (Samy)

    - Difficulté: Si on le fait en séparant chaque code binaire dans la string avec un espace et puis ici splittant la string, le fichier devient beaucoup trop grand. (Exemple, avec ici le fichier texte binaire de Frankenstein étant 5 fois plus grand que le fichier texte d’origine)

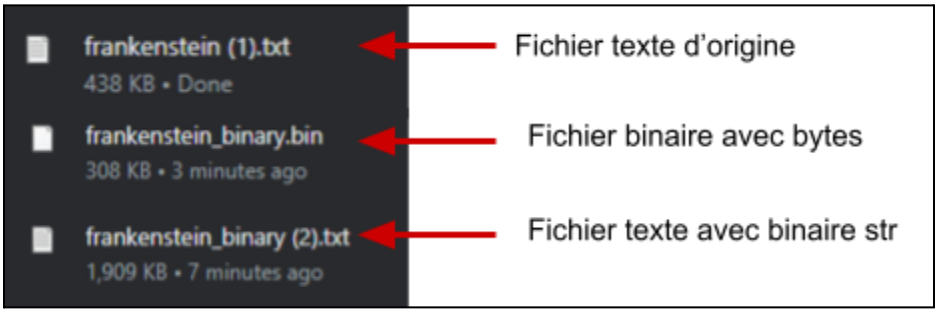


    - Solution: On ajoute chaque bit a une string “code” jusqu' à un code dans le dictionnaire, puis on ajoute chaque caractère trouvé à une string. (Samy)

- Modification de la fonction convert_text_binary(text) pour qu’il appelle toutes les fonctions nécessaires à l'encodage d’une fichier texte en fichier binaire et puis appelle toutes les fonctions nécessaires au décodage des bytes pour verifier que tout a été preservé. (Samy)

    - Difficulté: L'enregistrement du fichier .txt avec le binaire en tant que chaîne de caractères fait que le fichier est toujours trop grand, même sans les espaces

(Exemple, avec ici le fichier texte binaire de Frankenstein étant 4 fois plus grand que le fichier texte d'origine)



- Solution: Création de la fonction bits_to_bytes(binary_str): Coupe une string de bits en octets, ajoute les octets à une liste, et transforme chaque octet dans la liste en objet immutable bytes puis renvoie les bytes (Aide avec Python bytes()) (Samy)

- Puis write to fichier .bin contenant des bytes au lieu d'un fichier texte (Aide avec Python how to write to a binary file?) (Samy)

- Création de la fonction bytes_to_bits(binary): Format chaque byte sous la forme d'un octet et ajoute les octets à une string, puis renvoie la string de bits (Aide avec Converting integer to binary in python pour la fonction format()) (Samy)

    - Difficulté: Si la chaîne de bits sous forme de string n'est pas parfaitement divisible en huit, il manque des zéros et la restauration est erronée

Erreur: `nsi sinon rien non noi o`

    - Solution: Ajouter des zéros au début de la chaîne binaire str jusqu'à ce qu'elle soit un multiple de 8 (Samy)

Novembre 19, 2023 | 11/19/2023

Etapes

▶️ Création d'une interface pour facilement utiliser notre programme.

- Création de la fonction main() qui sert comme l'interface principale du programme: Permet à l'utilisateur de soit compresser soit décompresser un fichier texte, puis de choisir les fichiers en lui montrant les fichiers qui se situent dans le même répertoire que le code (Samy)

- Création de la fonction liste_fichiers(fin) pour trouver les fichiers dans le même répertoire que le code selon leurs fin (Aide avec Python: 5 Ways to List Files in Directory) (Samy)

```
Compression de textes avec l'encodage d'Huffman

1. Compresser un fichier texte
2. Décompresser un fichier texte

Entrez votre choix (1/2) : 1

Fichiers .txt disponibles :

1. frankenstein.txt
2. frankenstein_restored.txt
3. test.txt
4. test_restored.txt

Entrez l'indice du fichier à compresser : 3
```

- Ajout dans la fonction construction_arbre(phrase) de l'affichage de chaque étape de manière lisible en montrant le noeud créé ainsi que l'état de la liste (Samy)

```
Etape n 1 :

Noeud créé : [3, Cote Gauche: ('r', 1), Cote Droite: ('s', 2)]

Liste : [('e', 2), (<__main__.Noeud object at 0x7fd4eb327940>, 3), ('i', 3), ('o', 3), (' ', 4), ('n', 8)]

Etape n 2 :

Noeud créé : [5, Cote Gauche: ('e', 2), Cote Droite: [3, Cote Gauche: ('r', 1), Cote Droite: ('s', 2)]]

Liste : [('i', 3), ('o', 3), (' ', 4), (<__main__.Noeud object at 0x7fd4eb3278b0>, 5), ('n', 8)]
```

- Ajout dans la fonction convert_text_binary(text) de l'affichage de la liste des fréquences des caractères dans le texte, l'arbre finale, et le dictionnaire final avec les codes de chaque caractères (Samy)

```
Liste des caractères:

 [('r', 1), ('s', 2), ('e', 2), ('i', 3), ('o', 3), (' ', 4), ('n', 8)]

Arbre final:

 [23, Cote Gauche: [9, Cote Gauche: (' ', 4), Cote Droite: [5, Cote Gauche: ('e', 2), Cote Droite: [3, Cote Gauche: ('
r', 1), Cote Droite: ('s', 2)]]], Cote Droite: [14, Cote Gauche: [6, Cote Gauche: ('i', 3), Cote Droite: ('o', 3)], Co
te Droite: ('n', 8)]]

Codages des caractères:

 {'r': '0110', 's': '0111', 'e': '010', 'i': '100', 'o': '101', ' ': '00', 'n': '11'}
```

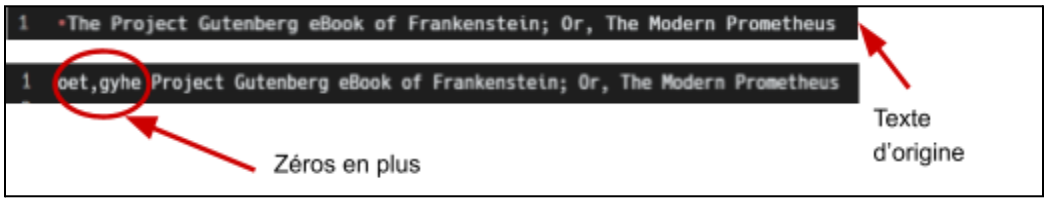
- Création de la fonction convert_binary_text(binaire, codes_csv) pour séparer la décompression de la fonction convert_text_binary(text) (Aide avec Reading a binary file with python)

Novembre 20, 2023 | 11/20/2023

Etapes

▶️ Lire un fichier .txt et le transformer en fichier .bin avec fichier .csv.
▶️ Transformer le fichier .bin avec le fichier .csv en nouveau fichier .txt.
▶️ Vérification de la taille des fichiers.

- Difficulté: Le tout début des textes est incorrect, car les zéros ajoutés pour rendre le binaire divisible par 8 signifient que les zéros du début ont été interprétés comme leurs propres codes.



- Solution prévue: Il faut inclure dans le csv le numéro des zéros ajoutés pour les ignorer lors du décodage. Dans le fichier .csv contenant un dictionnaire avec le codage des caractères, nous devons inclure le nombre de zéros ajoutés au début de la chaîne binaire afin d'en faire un multiple de 8. Ensuite, lorsque nous traversons notre string binaire pour la transformer en caractères UTF-8, nous devons d'abord ignorer les zéros du début.

- Modification de la fonction convert_text_binary(text) pour pouvoir écrire à un fichier .csv les valeurs du dictionnaire 'codes' qui servira à décoder le string binaire (Tony)

- Modification de la fonction convert_binary_text(binaire, codes_csv) pour pouvoir ouvrir un fichier .csv et ajouter ses valeurs à un dictionnaire ‘codes’ qui servira à décoder le string binaire (Tony)

  (Aide avec Stack Overflow pour la gestion des fichiers .csv)

- Ajout dans la fonction convert_text_binary(text) d’une colonne dans le dictionnaire ‘codes’ avec le nombre de zéros ajoutés à la string binaire et modification de la fonction phrase_lettres(binaire, codes) pour enlever les zéros ajoutés à la string binaire avant de la décoder (Samy)

- Création de la fonction phrase_to_bin(text) et ajout au menu dans la fonction main() d’un troisième choix qui permet de rentrer une string au lieu de seulement un fichier .txt (Tony)

- Ajout à la fonction convert_text_binary(text) qui trouve la taille des fichiers .bin créé et .txt d’origine et comparer la taille des deux (Samy et Tony)

Novembre 22, 2023 | 11/22/2023

- Difficulté: Par contre, le programme ne semble pas fonctionner sur Thonny avec des fichiers .txt, à cause d’une erreur de lecture de fichiers. Néanmoins, il est toujours utilisable sur Thonny si on utilise le troisième choix ou il prend une string dont on écrit directement et non un fichier texte.

```
Entrez l'indice du fichier à compresser : 8
Traceback (most recent call last):
  File "C:\Users\Samy\Downloads\Encodage Huffman.py", line 333, in <module>
    main()
  File "C:\Users\Samy\Downloads\Encodage Huffman.py", line 296, in main
    convert_text_binary(fichier)
  File "C:\Users\Samy\Downloads\Encodage Huffman.py", line 182, in convert_text_binary
    text_to_str = f.read()
  File "C:\Users\Samy\AppData\Local\Programs\Thonny\lib\encodings\cp1252.py", line 23, in decode
    return codecs.charmap_decode(input,self.errors,decoding_table)[0]
UnicodeDecodeError: 'charmap' codec can't decode byte 0x9d in position 13419: character maps to <undefined>
```

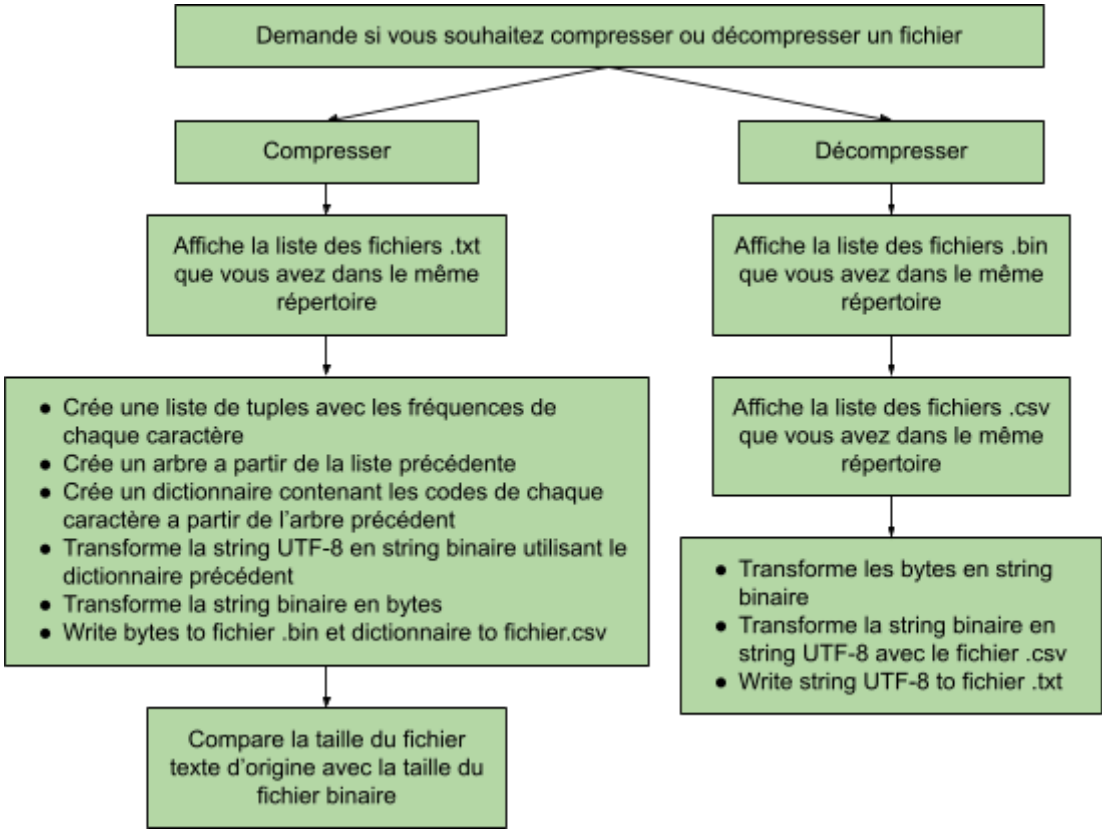
```
def convert_text_binary(text):
    #https://www.askpython.com/python/examples/read-file-as-string-in-python
    with open(text) as f:
        text_to_str = f.read()
```

- Solution: Avec Thonny, il faut spécifier l’encodage en UTF-8 quand on ouvre un fichier texte. (Samy)

```python
def convert_text_binary(text):
    #https://www.askpython.com/python/examples/read-file-as-string-in-python
    with open(text, "r", encoding="utf-8") as f:
        text_to_str = f.read()
```

Complétion du projet

Nous avons fait tout ce que nous avions prévu de faire et nous avons réalisé toutes les principales fonctionnalités de notre projet, comme le montre le graphique ci-dessous :



Vérification du projet

Avec le texte *"nsi sinon rien non none*," le fichier binaire fait 34% de sa taille originale, faisant 8 bytes au lieu de 23 bytes. En regardant les fichiers .txt d'origine et reconstitué, on voit que les deux sont exactement les mêmes.

```
Fichier binaire saved to test_binary.bin
Fichier csv saved to test_csv.csv

Taille du fichier texte : 23 bytes => 0 megabytes
Taille du bicher binaire: 8 bytes => 0 megabytes

La taille du fichier binaire est 0.34782608695652173 fois le texte d'origine.
```

Même avec le texte de Frankenstein de Mary Shelley (The Project Gutenberg eBook of Frankenstein, by Mary Wollstonecraft Shelley) possédant environ 7740 lignes, on observe que le fichier binaire fait 54% de la taille du fichier d'origine.

```
Fichier binaire saved to frankenstein_binary.bin
Fichier csv saved to frankenstein_csv.csv

Taille du fichier texte : 448965 bytes => 438 megabytes
Taille du bicher binaire: 244340 bytes => 238 megabytes

La taille du fichier binaire est 0.5442295056407515 fois le texte d'origine.
```