1. Justification de l'utilisation des modules :

Pour le projet, les modules bson et bitstring ne sont pas inclus dans python 3.6 , on les a donc mis dans le dossier « modules » .

-bson :

Dérivé de json permettant d'encoder un objet python en bytes. On ne peut pas utiliser la bibliothèque json car le module json ne produit pas d'objets bytes ou binary string.

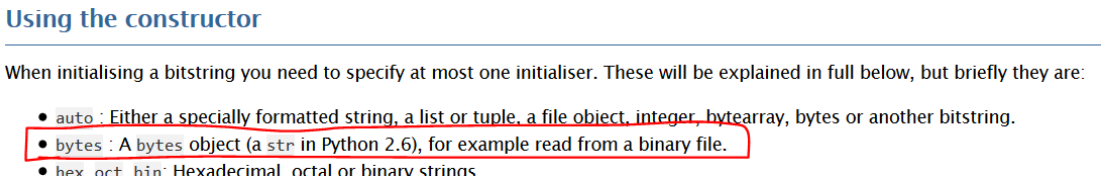
« The json module always produces str objects, not bytes objects. Therefore, fp.write() must support str input. » (https://docs.python.org/3/library/json.html, 19.2.1. Basic Usage)

Pour notre utilisation, on s'inspire d'un exemple : https://github.com/py-bson/bson

-bitstring :

D'après la documentation : https://pythonhosted.org/bitstring/contents.html

On peut en effet utiliser un objet bytes : https://pythonhosted.org/bitstring/creation.html



Using the constructor

When initialising a bitstring you need to specify at most one initialiser. These will be explained in full below, but briefly they are:

- `auto` : Either a specially formatted string, a list or tuple, a file object, integer, bytearray, bytes or another bitstring.
- `bytes` : A `bytes` object (a `str` in Python 2.6), for example read from a binary file.
- `hex, oct, bin`: Hexadecimal, octal or binary strings

Grace à ces deux bibliothèques on peut transformer un dictionnaire en chaine binaire et inversement. Voir « exemple.py » à la racine du projet.

-base64 :

Ce module permet d'encoder des binary string en text string et inversement. https://docs.python.org/2/library/base64.html

Dans notre cas, on veut compresser des fichiers autres que des fichiers texte mais on est limité par notre algorithme à compresser des chaines de caractères. On peut alors utiliser ce module pour y remédier.

-threading :

Quand on veut utiliser une interface graphique et exécuter un code en parallèle, l'interface se fige. On peut répartir les ressources en plusieurs threads qui s'exécuterons en même temps.

Pour le projet, on s'est appuyé sur un exemple proche de notre problématique. https://openclassrooms.com/forum/sujet/tkinter-et-threading

2. Optimisations

Le premier gain de temps a été fait sur la réécriture de la fonction occurrences(). Pour un fichier de 2,8Mo son temps d'exécution est passé de 2,3 secondes à 0,5 secondes.

Le second gain de temps significatif est sur la transformation de la chaine de caractère de 0 et de 1 finale ('0010100……) en Bytes. Pour un fichier de 194ko le temps total de compression passe de 54,5 secondes à 0,31 seconde.

Pour des fichiers plus lourds ces optimisations sont obligatoires sinon le temps d'exécution total devient très long.