TP D'INFORMATIQUE N°3
Dictionnaires : applications et implémentation

1 Compression d'un fichier texte

L'objectif de cette partie est d'écrire un programme permettant de *compresser* un fichier texte, c'est-à-dire de générer un fichier de taille moindre et pouvant être décompressé pour retrouver le contenu du fichier initial. Nous supposerons, pour simplifier l'exercice, que le fichier texte à compresser contient des mots séparés par des espaces, sans ponctuation.
Afin d'effectuer cette compression, nous allons numéroter chaque mot, pour pouvoir ensuite remplacer les mots par leur numéro. Pour conserver en mémoire quel numéro est associé à un mot donné, nous allons utiliser un dictionnaire D dont les clés sont des chaînes de caractères et les valeurs des entiers. Dans l'autre sens, pour retrouver le mot à partir du numéro, nous utiliserons un tableau de chaînes de caractères T. Le fichier compressé contiendra alors le tableau T (représenté par la suite de ses éléments, séparés par des espaces), puis le contenu du fichier initial, dans lequel les mots sont remplacés par leur numéro.
Par exemple, le texte :

> L objectif de cette partie est d écrire un programme permettant de compresser un fichier texte c est à dire de générer un fichier de taille moindre et pouvant être décompressé pour retrouver le contenu du fichier initial

sera compressé en :

> L objectif de cette partie est d écrire un programme permettant compresser fichier texte c à dire générer taille moindre et pouvant être décompressé pour retrouver le contenu du initial
> 0 1 2 3 4 5 6 7 8 9 10 2 11 8 12 13 14 5 15 16 2 17 8 12 2 18 19 20 21 22 23 24 25 26 27 28 12 29

Comme on le voit sur cet exemple, le fichier compressé peut être plus long que le fichier initial. Plus le texte contiendra de répétitions de mots, plus la compression sera efficace.

1. Donner le dictionnaire D, le tableau T et le texte compressé correspondants au texte :

   > Si six scies scient six cyprès six cents scies scient six cents cyprès

2. Écrire une fonction `numerotation` prenant en argument un texte sous forme de liste de mots, et renvoyant D et T tels que définis précédemment.
3. Écrire une fonction `phrase` prenant en argument un texte sous forme de liste de mots, et renvoyer le même texte sous forme de chaîne de caractères, où les mots sont séparés par des espaces.
4. Écrire une fonction `compresser` prenant en argument le nom du fichier source et le nom du fichier compressé à créer, et procédant à cette compression.

   On redonne les fonctions suivantes pour la lecture et écriture de fichier :
   - `f=open('nom_du_fichier.txt','r')` permet d'ouvrir un fichier en lecture (*read*), sous le nom de variable `f`. `f=open('nom_du_fichier.txt','w')` permet d'ouvrir un fichier en écriture (*write*), sous le nom de variable `f`. Le fichier sera créé s'il n'existe pas déjà.
   - `f.readlines()` renvoie la liste des lignes présentes dans le fichier `f`, chaque ligne étant représentée sous forme de chaîne de caractères.
   - `ch.strip()` renvoie la chaîne de caractère obtenue à partir de la chaîne `ch` en supprimant tous les caractères spéciaux, tels que les retours à la ligne.
   - `ch.split(' ')` renvoie la liste des mots séparés par une espace apparaissant dans la chaîne `ch`.
   - `f.write(ch)` ajoute dans le fichier `f` ouvert en écriture la chaîne de caractères `ch`. `\n` au sein d'une chaîne provoque un retour à la ligne. La fonction `str` permet de transformer un entier en chaîne.
   - `f.close()` ferme le fichier `f` (ce qu'il faut toujours faire une fois qu'on a fini de travailler sur un fichier qu'on a ouvert).

5. Télécharger le fichier `texte.txt` sur cahier-de-prepa et le compresser. Le fichier compressé est-il moins lourd en mémoire ?
6. Écrire une fonction `decompresser`
7. Décompresser le fichier compressé obtenu précédemment. Retrouve-t-on bien le texte initial ?

2 Tables de hachage

L'objectif de cette partie est d'implémenter une structure de dictionnaire par table de hachage. On ne considérera que des dictionnaires dont les clés sont des chaînes de caractères.

1. la fonction `ord` permet d'obtenir l'entier associé à un caractère. Écrire une fonction `hash` prenant en argument une chaîne `ch` et un entier n, et renvoyant la somme des entiers associés aux caractères de `ch`, modulo n.
2. À l'aide de cette fonction de hachage, et en représentant un dictionnaire par un tableau de listes d'associations, implémenter les opérations suivantes :
   - `creer_dico(n)`, renvoyant un dictionnaire vide contenant n alvéoles ;
   - `cle_presente(D,cle)`, testant si une clé est présente dans un dictionnaire ;
   - `ajouter(D,cle,valeur)`, ajoutant un couple clé, valeur à un dictionnaire ;
   - `valeur_associee(D,cle)`, renvoyant la valeur associée à une clé ;
   - `modifier_valeur(D,cle,valeur)`, associant une nouvelle valeur à une clé déjà présente ;
   - `supprimer(D,cle)`, supprimant une clé, et sa valeur associée.
3. Modifier les fonctions de la partie précédente pour qu'elles utilisent cette implémentation des dictionnaires plutôt que l'implémentation native de Python. On prendra pour n la longueur de la liste de mots à numéroter. Vérifier que le résultat de la compression et décompression est inchangé.
4. Une bonne fonction de hachage doit limiter le nombre de collisions. Écrire une fonction `nb_collisions` prenant en argument un dictionnaire, et comptant sa fréquence de collisions, c'est-à-dire le nombre de clés qui ne sont pas seules dans leur alvéole, divisé par le nombre total de clés. Calculer la fréquence de collisions du dictionnaire obtenu par la compression du fichier `texte.txt`.
5. Écrire une nouvelle fonction de hachage, calculant pour une chaîne $c_0, \dots c_k$ l'entier

$$\left(\sum_{i=0}^{k} \text{ord}(c_i) 101^i\right) \%n$$

   Cette fonction de hachage génère-t-elle moins de collisions que la précédente ?