Compression par dictionnaire : LZ78

1 Présentation de la méthode

On pourra se référer à `https://members.loria.fr/EJeandel/files/Codage/02-1.pdf`, et une version plus longue du même auteur : `https://members.loria.fr/EJeandel/files/Codage/02.pdf`.

L'idée est :

— De constituer un dictionnaire de mots déjà rencontrés.

— A chaque itération, on cherche le premier mot en partant du début non présent dans le dictionnaire.

Ce mot est de la forme :

$$a_0 a_1 \cdots a_{n-1} a_n$$

tel que $a_0 a_1 \cdots a_{n-1}$ est dans le dictionnaire, mais pas $a_0 a_1 \cdots a_{n-1} a_n$. Soit i le numéro de $a_0 a_1 \cdots a_{n-1}$ dans le dictionnaire.

On ajoute alors ce nouveau mot $a_0 a_1 \cdots a_{n-1} a_n$ au dictionnaire ; et on peut le coder comme (i, a_n) signifiant que le mot suivant est le mot numéro i, suivi de la lettre a_n.

2 Implémentation

Compression : écrire une fonction `compression(texte)` prenant une chaîne de caractères et renvoyant la liste des couples codant ce texte.

```
>>> compression('ABRABADABRA')
[(0, 'A'), (0, 'B'), (0, 'R'), (1, 'B'), (1, 'D'), (4, 'R'), (0, 'A')]
```

Cette fonction remplira au fur et à mesure un dictionnaire renvoyant, pour chaque chaîne rencontrée, son indice. La première entrée du dictionnaire pourra être `"" : 0`.

```
d = {'' : 0}
```

Dès qu'un nouveau mot sera rencontré, il faudra l'ajouter au dictionnaire, et ajouter ce qu'il faut à la séquence compressée.

Décompression : écrire une fonction `decompression(texte)` prenant une séquence de couples correspondant à la compression d'un texte et retournant le texte décompressé.

```
>>> decompression([(0, 'A'), (0, 'B'), (0, 'R'), (1, 'B'), (1, 'D'), (4, 'R'), (0, 'A')])
'ABRABADABRA'
```

Cette fonction constituera le même "dictionnaire" que la fonction de codage au fil de la décompression. Mais dans l'opération de décompression, on cherchera à passer de l'indice au mot, et non du mot à l'indice. On pourra donc travailler avec une simple liste, initialisable par :

```
d = ['']
```

Pour chaque couple de la séquence compressée, on calculera le mot codé, que l'on ajoutera au dictionnaire et à la séquence décompressée.

Lecture d'un fichier : lire le contenu d'un fichier texte, (par exemple `germinal.txt`, qui est en caractères ascii), et l'enregistrer dans une chaîne de caractères. Tenter la compression puis décompression sur cette chaîne, et vérifier que l'on

Taux de compression : comparer la longueur du texte en entrée, et la longueur de la séquence compressée. Calculer le taux de compression que l'on pourrait avoir en supposant que le texte est codé en caractères ascii et en choisissant un codage de la séquence compressée.

3 Pour aller plus loin : écriture de la version compressée dans un fichier.

Une fois calculée la séquence compressée, il faut l'écrire dans un fichier qui sera la version compressée du fichier d'origine.

En reprenant l'exemple de la version compressée de ABRABADABRA :

$$(0, A)(0, B)(0, R)(1, B)(1, D)(4, R)(0, A)$$

3.1 Version très naïve

on pourra écrire dans un fichier texte :

```
A0
B0
R0
B1
R4
A0
```

Le caractère de fin de ligne `\n` sert ici de séparateur entre chaque couple, pour rendre le codage plus lisible. En général, il sera nécessaire de choisir un caractère séparateur non présent dans le texte : un caractère ascii non utilisé.

Avec cette version, le fichier compressé sera plus gros que le fichier d'origine : chaque caractère, codé en ascii, occupe un octet ; l'écriture de chaque chiffre d'un nombre comme un caractère ascii est très peu efficace...

3.2 Pour aller plus loin

Pour avoir une version réellement efficace, il faut écrire bit à bit.

On peut calculer combien de bits sont nécessaires pour coder chaque nombre index du dictionnaire : si n est la taille du dictionnaire, ce nombre s'écrit en $\lceil \log_2(n) \rceil$ bits. On peut ainsi écrire chaque couple en codant la lettre par son code ascii, suivi du nombre en exactement $\lceil \log_2(n) \rceil$ bits.

Il n'y a alors pas besoin de séparateur entre les couples, puisque l'on connaît leur taille. Il sera par contre nécessaire de préciser, en début de fichier compressé, ce nombre de bits $\lceil \log_2(n) \rceil$ codant chaque nombre pour permettre le décodage.

Une version un peu plus optimisée est expliquée ici `http://defeo.lu/in420/dm-lz78` : en réalité, le couple i fait référence à une entrée du dictionnaire de numéro inférieure à i. On peut donc écrire ce nombre en $\lceil \log_2(i) \rceil$ bits. On fait ainsi grossir le nombre de bits nécessaire à coder chaque couple au fur et à mesure.