Université Côte d'Azur
Automates & Langages

L3 Info, L3 MI, L3 ST
2023–24

TP n°5

Compression de texte

Présentation

La compression de texte consiste à réduire la taille d'un fichier binaire en vue de faciliter sa transmission ou diminuer son coût de stockage. Elle doit garantir qu'à partir du fichier *compressé*, on puisse reconstituer le fichier d'origine. Il s'agit donc de compression *conservative* ou *sans perte*.

Aujourd'hui, on se propose d'implanter l'algorithme de Lempel-Ziv-Welsh datant de 1984. C'est une amélioration de l'algorithme d'origine de Lempel-Ziv de 1977 et aussi de son successeur de 78, respectivement baptisés LZ77 et LZ78. L'algorithme LZW est celui du format de compression de texte `zip` ou des formats d'image `gif` ou `tiff`.

Ces méthodes de compression peuvent être exécutées à la volée, par opposition à celles qui requièrent au préalable une étude statistique du texte ou de l'alphabet sur lequel est écrit le texte. Toutefois, cette méthode utilise un *dictionnaire* qui s'allonge au fur et à mesure que la compression s'effectue.

Algorithme LZW

Principe

Le principe de l'algorithme est le suivant. Au départ, le dictionnaire contient un jeu de caractères c'est-à-dire seulement des chaînes de longueur 1. A chaque caractère du texte lu, on consulte le dictionnaire pour savoir si le préfixe courant du texte est dans le dictionnaire ou pas, et sinon, on l'y rajoute. La compression réside dans la liste des index de chaque facteur lu. Bien sûr, l'insertion dans le dictionnaire mériterait d'être optimisée pour que l'algorithme soit efficace.

Préliminaires

On commence par écrire 3 petites fonctions de base pour la création et la gestion du dictionnaire.

Exercice 1.

1. Ecrivez une fonction `creeDT(n)` qui à partir d'un entier n retourne un couple (d, t) où d est le dictionnaire à proprement parler et t sa taille effective *i.e.* son nombre d'éléments significatifs. Le dictionnaire LZW sera ici modélisé par une liste contenant les 95 caractères UNICODE allant du 32e (' ') au 126e ($\sim$). Les n cases suivantes seront initialisées au mot vide.
2. Ecrivez une fonction `estPlein()` qui teste si le dictionnaire d extrait du couple (d, t) est plein.
3. Ecrivez une fonction `ajoute(dt,ch)` qui ajoute la chaîne ch au dictionnaire d du couple $dt = (d, t)$ à condition qu'il ne soit pas déjà plein. Cette fonction ne sera appelée qu'en l'absence du mot, donc inutile de se soucier de sa présence ou non dans le dictionnaire.

   Ainsi, à partir du couple dt et de la chaîne ch, elle retourne le couple dt réactualisé ou inchangé selon le cas.

Codage ou compression

L'algorithme LZW consiste à mettre à jour le dictionnaire des facteurs tout en produisant une liste d'index qui tient lieu de compression. Pour un mot m donné du dictionnaire, tous ses préfixes devront appartenir au dictionnaire. A chaque instant, on se base sur le facteur courant m du texte qui appartient déjà au dictionnaire. Puis on lit la lettre suivante a. Deux cas se présentent :

- soit le mot ma n'est pas dans le dictionnaire : l'index de m est mis dans la liste-résultat, le facteur ma est rajouté au dictionnaire et le mot courant devient $m = a$;
- soit le mot ma appartient au dictionnaire et le mot courant devient $m = ma$.

Exercice 2.

1. Déroulez *à la main* l'algorithme sur l'exemple `'abracadabra'` pour bien le comprendre. Le mot courant est initialisé à la première lettre du texte.
2. Ecrivez une fonction `encodeLZW()` qui prend un texte en paramètre et qui le compresse selon cet algorithme.

Décodage ou décompression

Le décodage consiste à lire un par un les élements de la liste contenant un texte compressé et à restituer le facteur correspondant dans le dictionnaire. Cela suppose que le dictionnaire est recréé à la volée, l'algorithme pourrait être le suivant :

0. le 1er index de la liste permet de décoder le 1er facteur m correspondant dans le dictionnaire ;
1. l'index suivant de la liste permet d'écrire sur la sortie le facteur mot correspondant dans le dictionnaire ;
2. on ajoute éventuellement au dictionnaire le mot m augmenté de la première lettre du facteur suivant mot ;
3. m prend la valeur de mot et on reprend en 1.

Presque ! Cela ne marche pas car il y a un cas problématique à traiter à part. C'est celui où on lit l'index d'un mot qui n'est pas encore dans le dictionnaire ...

Exercice 3.

1. Décodez *à la main* le compressé $[33, 34, 95, 97, 35]$ afin de bien comprendre l'algorithme en général et aussi le cas particulier qui peut se produire.
2. Ecrivez la fonction `decodeLZW()` qui restitue un texte à partir de la liste d'index qui résulte de la compression dans le cas général. L'algorithme est celui décrit plus haut, vous pouvez le tester sur l'encodage du mot `'abracadabra'` par exemple.
3. Dans le cas particulier, on a recours à un `try` ... `except` ... . Il permet d'éviter l'appel à un élément du dictionnaire qui n'a pas encore été entré. Dans ce cas, on peut tout de même décoder puisque si m est le dernier mot décompressé, le mot à restituer est $mm[0]$. Cela va coïncider avec l'ajout de ce même mot au dictionnaire. Terminez à présent la fonction `decodeLZW()`.

Exercice 4.

Maintenant que les fonctions de codage/compression et de décodage/décompression sont écrites, mettez-les en œuvre sur des fichiers ou jeu de caractères plus conséquents.