Projet d'algorithmique

Algorithme de compression par déplacement en tête de liste

Objectif: On souhaite implanter une méthode d'encodage et de compression de données (adaptée aux textes) qui utilise une liste chaînée de mots. Dans la suite, la position d'un mot dans la liste est un entier strictement positif (la position du mot en tête de liste est donc 1). Un mot est une suite de moins de 128 caractères. On ne demande pas de gérer le cas où un mot plus long apparaît.

1 Le codage des mots

Pour chaque mot du texte
- Rechercher le mot dans la liste.
- Si le mot existe dans la liste, on le code à l'aide de sa position dans la liste, puis on le déplace en tête de liste.
- Si le mot n'est pas dans la liste, on le code par l'entier `0` suivi du mot en clair, et on ajoute ce mot en fin de liste.

Exemple (on ignore ici les séparateurs et la ponctuation)

On considère le texte suivant :

deux et deux quatre
quatre et quatre huit
huit et huit seize
répétez dit le maître

1. On code *deux* par `0 deux`. La liste est *deux*.
2. On code *et* par `0 et`. La liste devient *deux*→*et*.
3. On code *deux* par `1` car il est en première position de la liste. La liste n'est pas transformée car *deux* est déjà en tête de liste.
4. On code *quatre* par `0 quatre` puisqu'il n'est pas dans la liste. La liste devient *deux*→*et*→*quatre*.
5. On code le *quatre* suivant par `3`. La liste devient *quatre*→*deux*→*et*.
6. On code le *et* suivant par `3`. La liste devient *et*→*quatre*→*deux*.
7. On code le *quatre* suivant par `2`. La liste devient *quatre*→*et*→*deux*.
8. On code le *huit* par `0 huit`. La liste devient donc *quatre*→*et*→*deux*→ *huit*.
9. Les deux premiers vers sont donc codés par : `0 deux 0 et 1 0 quatre 3 3 2 0 huit`.

Cette représentation encodée peut être écrite dans un fichier sous la forme de chaînes de caractères séparées par des espaces.

2 Décodage

Au décodage on peut lire chaque chaîne de caractères dans le fichier précédent et reconstruire la liste en l'utilisant de manière identique à ce qu'a effectué l'encodeur.

- Si on lit `0` , on lit le mot suivant, on l'écrit, on l'ajoute en fin de liste.
- Si on lit un entier `n` non nul, on extrait la cellule en position `n`, on écrit son contenu, on la déplace en tête de liste.

3 Ponctuation et mise en forme

Dans l'exemple donné ci-dessus, la ponctuation et la mise en forme (retour à la ligne) ont été ignorées. On considère désormais qu'un texte est formé alternativement de mots courants, formés de caractères alphabétiques, et de mots de ponctuation, formés de caractères non alphabétiques.

L'encodeur et le décodeur vont donc utiliser alternativement deux listes : celle des mots alphabétiques et celle des mots de ponctuation.

De plus, afin de reconnaitre plus facilement la fin des mots, on choisit de faire précéder les mots en clair par leur longueur.

Reprenons les deux premiers vers de notre exemple en leur ajoutant des symboles de ponctuation ou de mise en forme (dans l'exemple ci-dessous, le symbole ↩ sert à visualiser les caractères non-alphabétiques du retour à la ligne [1], et le symbole _ sert à visualiser le caractère espace du texte initial qu'on cherche à encoder) :

deux, et deux, quatre ↩
quatre, et quatre, huit.

1. On code *"deux"* par `0 4 deux`. La liste des mots est *"deux"*
2. On code *", "* par `0 2 _`. La liste des ponctuations est *",_ "*
3. On code *"et"* par `0 2 et`. La liste des mots devient *"deux"*→*"et"*
4. On code l'espace *" "* par `0 1 _`. La liste des ponctuations devient *",_ "*→*"_ "*
5. On code *"deux"* par `1` car il est en première position de la liste des mots. La liste n'est pas transformée car *deux* est déjà en tête de liste
6. On code *", "* par `1` car il est en première position de la liste des ponctuations. Cette liste n'est pas transformée car *", "* est déjà en tête de liste
7. On code *"quatre"* par `0 6 quatre` puisqu'il n'est pas dans la liste des mots. Cette liste devient *"deux"*→*"et"*→*"quatre"*
8. On code le retour à la ligne "↩" par `0 1 ↩`. La liste des ponctuations devient *",_ "*→*"_ "*→*"↩ "*
9. On code le *"quatre"* suivant par `3`. La liste devient *"quatre"*→*"deux"*→*"et"*
10. On code *", "* par `1` car il est en première position de la liste des ponctuations. Cette liste n'est pas transformée car *", "* est déjà en tête de liste
11. On code le *"et"* suivant par `3`. La liste devient *"et"*→*"quatre"*→*"deux"*
12. On code l'espace suivant par `2`. La liste des ponctuations devient *"_ "*→*",_ "*→*"↩ "*
13. On code le *"quatre"* suivant par `2`. La liste devient *"quatre"*→*"et"*→*"deux"*
14. On code *", "* par `2`. La liste des ponctuations devient *",_ "*→*"_ "*→*"↩ "*

1. Dans cet exemple, on a choisi de représenter la fin de ligne par un seul caractère, mais en général elle est représentée par les caractères non-alphabétiques de code ASCII 13 (Carriage return, `'\r'`) et 10 (Line Feed, `'\n'`).

15. On code le *huit* par `0 huit` dans la liste des mots. La liste devient donc *"quatre"→"et"→"deux"→"huit"*
16. On code le dernier point *"."* par `0 1 .` dans la liste des ponctuations. Cette liste devient *",_ "→"_ "→"↩"→"."*

Ces deux vers sont donc codés par : `0 4 deux` `0 2 ,_` `0 2 et` `0 1 _` `1` `1` `0 6 quatre` `0 1 ↩` `3` `1` `3` `2` `2` `2` `0 4 huit` `0 1 .`

Là encore, cette représentation encodée peut être écrite dans un fichier sous la forme de chaînes de caractères, et le décodage fonctionne comme précédemment, en alternant l'usage des deux listes.

4 Compression

Dans les méthodes décrites ci-dessus, où l'on représente les mots, les positions et les longeurs sous la forme de chaînes de caractères séparées par des espaces, de nombreux espaces inutiles sont utilisés et les entiers sont écrits comme des chaînes de caractères.

Puisqu'on a maintenant la longueur des mots, on peut supprimer l'espace séparateur à la fin de chaque mot.

De plus, comme on sait que les mots font moins de 128 caractères, on peut représenter leur longueur en binaire sur un octet plutôt que sous la forme d'une chaîne de caractères. Enfin, de la même manière, on pourra considérer que la position d'un mot peut être représentée en binaire sur deux octets (un `short`).

5 Ce qui est demandé

Dans votre développement, il vous est conseillé de procéder par degré de difficulté en commençant par gérer les cas sans ponctuation ni mise en forme, puis en prenant en compte la ponctuation et la mise en forme en encodant sous la forme de chaînes de caractères (ce qui vous permettra de "lire" facilement dans le fichier produit ce que vous avez encodé, avant de terminer par la représenation binaire des entiers et la suppression des espaces. À chaque étape, vous pourrez vérifier que après encodage et décodage vous retombez bien sur les données initiales.

Écrivez un programme `Compresse -[cd] entree sortie`.
Avec l'option `-c` le programme effectue la compression/encodage du fichier `entree` et écrira le résultat dans le fichier `sortie`, avec l'option `-d` il effectuera la décompression/décodage du contenu du fichier `entree` et écrira le résultat dans le fichier `sortie`.

Le projet est à réaliser en binôme. Une archive de nom `login1_login2.tar` où `login1` et `login2` sont les logins des deux membres du binôme, contenant les sources (makefile inclus) et un rapport en pdf devra être déposée sur le compte `elearning.univ-mlv.fr` **des deux membres du binômes**. Une soutenance pourra être organisée.

6 Remarque

L'intérêt de la méthode est qu'un mot fréquent est souvent en tête de liste et est donc codé par un entier petit dont l'écriture nécessite peu de bits. La dernière méthode décrite utilise deux octet pour la position de chaque mot, mais on peut faire mieux avec un nombre d'octets variables, voir un nombre de bits variables... pour ceux à qui il resterait du temps libre !