PLS - LZW Compressor and Decompressor - Compte-Rendu

Vous trouverez dans ce fichier le compte-rendu pour le projet de PLS, réalisé par :

- HALLAL Marwan
- NOGUERON Matthieu
- REVEL Antoine
- VIAL-GRELIER Aymeric

1) Objectif de ce projet

Ce projet a pour objectif de nous faire coder une application de compressage et décompressage, selon la méthode **Lempel-Ziv-Welch.**

Nous serons amené à travers celui-ci, à travailler sur des fichiers binaires, à lire des fichiers octets par octets, mais aussi à travailler avec des représentations de données sur plus de 8 bits. De plus, pour la méthode LZW, l'utilisation d'un dictionnaire étant obligatoire, nous devrons définir la structure de notre dictionnaire. Afin d'avoir des coûts de recherche faible et une structure claire, nous devrons donc trouver une architecture cohérente et fiable pour notre projet.

Enfin l'ensemble de se projet se prêtera assez bien à une division équitable du travail à travers le groupe.

2) Utilisation de l'application

Récupération des sources du projet

Vous pourrez trouver l'ensemble des sources pour ce projet, ainsi que l'historique des modifications faites sur celui-ci sur notre GIT, en utilisant les commandes :

Pour une connexion en ssh :

```
git clone git@github.com:RICM/LZW\_Un\_Compressor.git
```

Pour une connexion en http :

```
git clone https://github.com/RICM/LZW\_Un\_Compressor.git
```

Compilation des sources

L'application et l'ensemble des exécutables se créent dans un dossier target, pour pouvoir faire un make correcte, ce dossier doit donc être présent dans le projet. S'il n'y est pas (ce qui est de base quand on récupère les sources depuis le GIT), il faudra le créer.

Pour compiler il suffira de faire :

```
make
```

Si on souhaite se limiter à la compilation de l'exécutable de compression décompression en se passant des exécutables de test, on pourra faire :

```
make lzw
```

Enfin si on veut supprimer l'ensemble des exécutables et les fichiers crées lors de la compilation, il suffira de lancer la commande suivante :

```
make clean
```

Utilisation de l'application

Pour utiliser lzw, en compression nous devrons utiliser la commande suivante :

```
./lzw -c [fileName] [archiveName]
```

Et pour la décompression :

```
./lzw -d [archiveName.lzw]
```

Pour utiliser RLE :

```
// compression
./lzw -cr [fileName] [archiveName]

// décompression
./lzw -dr [archiveName.lzw]
```

3) Séparation des modules

Afin de pouvoir travailler à plusieurs sur le même projet avec GIT, nous avons décidé de séparer le projet en différents modules :

- *binrw* : lecture et écriture en binaire, sur un certains nombre de bits
- *sequence* : implémentation de la structure de liste d'octets représentant une séquence de char
- *tree* : implémentation de la structure d'arbre utilisée pour notre dictionnaire
- *dictionary* : implémentation de la structure globale de dictionnaire et des fonctions de recherche dans un arbre
- *compression* : module contenant les fonctions de compression et de décompression LZW
- *rle* : module de compression rle
- *encapsulate* : module d'encapsulation de l'extension du fichier original dans le fichier compressé et désencapsulation à la décompression (pour ne pas avoir à donner d'extension)

- *bindump* : module supplémentaire de dump binaire, utilisant binrw et nous permettant de visualiser nos fichiers. Très utile lors des phases de débogage, nous avons décidé de le laisser dans le projet.

4) Possibilité d'évolution

Différentes améliorations pourront être menées sur ce projet. Tout d'abord, notre algorithme de compression et de décompression étant très long, nous pourrons améliorer ceci.

Nous nous sommes rendu compte que tardivement que lors de l'ajout d'un élément (w.c) dans notre dictionnaire, nous le reparcourons pour trouver où insérer l'élément, alors que notre code nous donne la possibilité de faire un insertion simple dans un arbre que nous avons sauvegardé précédement (résultat de la recherche de w). Cette amélioration pourrait drastiquement diminuer le temps d'exécution de la compression et de la décompression.

De plus, nous pourrions dans une version future ajouter des algorithmes de compression afin de mieux compression nos fichiers (tel que tar.gz qui utilise LZW, couplé à deux passage de Huffman). Une version avec move-to-front pourrait aussi être ajoutée.

Enfin, nous aurions pu créer une interface graphique sous GTK afin de rendre notre application accessible par des personnes ayant peu de connaissance en matière de ligne de commande.