

Compression de données (LAB2414 1/2)

Vous utiliserez la méthode du codage arithmétique, [vue en cours](#) (p.14-15), pour compresser des fichiers de texte. Votre code sera évalué en fonction de son efficacité de compression.

Votre programme prend un seul caractère en paramètre (« c » ompresse ou « d »écompresse) et utilise les entrées et sorties standard pour lire et écrire des données à compresser (ou décompresser).

Ainsi les tests de base s'effectueront comme ceci :

```
valgrind ./myCompress c < fichierEntree.txt > fichierComprimee.z
valgrind ./myCompress d < fichierEntree.z > fichierDecompresse.txt
cmp fichierEntree.txt fichierDecompresse.txt
```

Les symboles « > » et « < » servent à utiliser directement les flux d'entré/sortie standard (*stdin/stdout*). La première ligne lance la compression d'un fichier, la deuxième lance la décompression de ce fichier, et la troisième vérifie que le fichier original et le fichier décompressé sont bien exactement les même. L'outil `valgrind` vérifiera les erreurs et fuites mémoire qui vous coûterons de points. La fonction `cmp` nous permettra de vérifier que le fichier reconstitué est parfait, elle n'affiche rien si les 2 fichiers donnés en paramètres sont les mêmes, sinon elle affiche la position de la première différence.

Notation :

- fonctionnelle
 - la compression (transformation d'un fichier texte en un fichier binaire), 5 points
 - la décompression (récupération du fichier originel), 5 points
- pour pouvoir bénéficier de points au delà de 10, il faut que le fichier récupéré soit **exactement le même** , à l'octet près !
- Les points de 10 à 20 reflèterons l'efficacité de compression
- qualité de code (testée avec l'outil [valgrind](#))
 - Vous perdrez 1 point par fuite mémoire, même d'un seul octet. Vous en perdrez 2 si cela à lieu à compression et à la décompression.
 - Vous perdrez 2 points en cas d'erreur mémoire, vous en perdrez donc 4 si cela à lieu à compression et à la décompression.

Ces points pourront être retirés même si vous avez moins de 10 points !

Conseils : pour pouvoir le maximum de points pour votre programme il faut que vous réalisiez la compression, la décompression, et que la compression soit sans perte d'information. **Ainsi un superbe programme de compression de données, mais qui n'a pas la fonctionnalité de décompression, ou si le fichier décompressé n'est pas exactement le même que le fichier d'origine, cela amènera à une note <=10.** Donc veillez à d'abord avoir une compression/décompression qui marche parfaitement, et seulement ensuite vous essayez de l'améliorer. Pensez à tester entièrement (compression/décompression) votre programme à la moindre modification... Et bien sûr en utilisant l'outil *valgrind*.

Voici, de manière indicative, la décomposition des séances

Séance 1

Introduction et structure du programme:

- Apprendre à récupérer et traiter les arguments donné à un programme. Pour notre programme cela se résume à récupérer le premier argument et voir si il s'agit d'un « c » (demande de compression) ou d'un « d » (demande de décompression).
- Lire le flux *stdin*, le mettre en mémoire (fonction *fgetc*).
- Comptez la fréquence de chaque symboles du fichier à compresser.
- Créez le tableau d'intervalles utilisé par la méthode de codage arithmétique, et enregistrez le sur *stdout*, via la fonction *fwrite*.
- Créez le début de la décompression : lire (*fread*) le tableau d'intervalles et le mettre en mémoire.

Les 2 derniers points peuvent être laissés en travail personnel (fonctions, *fwrite*, *fread*) déjà vues en cours et TD. Mais tous ces points doivent absolument être opérationnels pour la séance 2!

Séance 2

Implémenter le codage/décodage arithmétique :

Conseil : pour commencer faites d'abord la compression/décompression sur un seul nombre flottant et identifiez les tests d'arrêt à mettre en place (à la compression et à la décompression) pour ne pas aller au delà de la précision des flottants...

Codage

1. Soit X une séquence de m symboles $X=x_1x_2\dots x_m$ qui prend ses valeurs à partir d'une source $S=\{s_1s_2\dots s_m\}$,
2. Calculer la probabilité associée à chaque symbole de la source S , notons $p_i=\text{Probabilité}(S=s_i)$
3. Associer à chaque symbole s_k un sous intervalle $[L_{S_k}, H_{S_k})$ proportionnel à sa probabilité, avec $L_{S_k} - H_{S_k} = p_k$.
4. Initialiser la limite inférieure L (Low) de l'intervalle de travail à la valeur 0 et la limite supérieure H (High) à la valeur 1.
5. Tant qu'il reste un symbole dans la séquence à coder, et que la largeur est suffisante (**)
(précision limitée)
 - largeur = $H - L$
 - $H = L + \text{largeur} * H_{S_k}$
 - $L = L + \text{largeur} * L_{S_k}$
6. A la fin, n'importe quelle valeur de l'intervalle $[L,H)$ représente d'une manière unique la séquence d'entrée (en général on utilise la valeur $(L+H)/2$).

(**) pour des flottants classique (codé sur 4 octets) et sur des exemples simples, une limite de largeur à 0.0001, semble fonctionner. À vous de vérifier cela en pratique sur les exemples de fichiers qui vous seront donnés.

Décodage

On lit les données compressées et on les met dans une valeur X.

1. D'abord on doit initialiser la limite inférieure (L) de l'intervalle de travail à la valeur 0 et la limite supérieure (H) à la valeur 1,
2. Tant qu'il reste un symbole à décoder : (et que la précision de largeur est suffisante (**))

$$\text{largeur} = H - L$$

Chercher le sous intervalle $[L_{S_k}, H_{S_k})$ du symbole à décoder sachant que :

$$L_{S_k} \leq (X - L) / \text{largeur} \leq H_{S_k}$$

3. décoder (et enregistrer) le symbole s_k
4. $H = L + \text{largeur} * H_{S_k}$
5. $L = L + \text{largeur} * L_{S_k}$

Généralisez votre codage à l'utilisation, et l'enregistrement, successif d'autant de valeurs flottantes nécessaire à la compression d'un texte complet.

Vous constaterez alors un (petit) problème : Lors de la constitution de la dernière valeur flottante, il n'y aura probablement plus assez de caractères pour « remplir entièrement le flottant », des symboles supplémentaires apparaîtront alors lors de la décompression... La solution est tout simplement d'enregistrer, au début du fichier compressé, de la taille exacte du fichier original, cela permettra alors d'arrêter la décompression au bon moment.

Effectuez des mesures d'efficacité de votre compression.

Séance 3

Amélioration de votre compression.

- Voir le codage « move to front » qui peut être implémenté en amont du codage arithmétique et améliore son efficacité.
- Utiliser $[-1;+1]$ au lieu de $[0;1]$, ce qui permet d'utiliser un bit en plus (le bit de signe des flottants).
- Utiliser des `double` à la place des `float`. (et donc changer la limite de précision, permettant de stocker plus d'information dans un même nombre.)

Mesurez de combien vous avez amélioré la méthode de base.

Attention testez bien, et entièrement (compression et décompression), toutes vos dernières améliorations avant de rendre votre code...

Bon courage.