INF4710
Introduction aux technologies multimédia

A2016 - Travail pratique #1
Compression d'images fixes

Objectifs :

- Permettre à l'étudiant de se (re)familiariser à la manipulation de données et au traitement d'images en C++, ou bien avec le logiciel Matlab
- Permettre à l'étudiant de s'initier aux algorithmes de compression de données sans perte (dans ce cas-ci, LZ77)

Remise du travail :

- Avant le 19 septembre 2016, 14h00, sur Moodle – **aucun retard accepté**

Références :

- Voir les notes de cours sur Moodle (Compression sans perte, p.40--44)

Documents à remettre :

- L'ensemble de votre code source (.m pour Matlab, .hpp/cpp pour C++, mais pas les deux!)
- Un rapport (**format .pdf**) contenant un survol bref de votre travail (présentation de l'algo et de votre implémentation, ~1 page), et un tableau de vos taux de compression pour toutes les images en plus de quelques lignes de discussion (~1 page)

Autres directives :

- Pensez à commenter l'ensemble de votre démarche directement dans votre code! Sinon, difficile d'attribuer des points lorsque ça ne fonctionne pas...
- Les TD s'effectuent **obligatoirement** en équipe de deux personnes (peu importe la section de laboratoire). Utilisez le forum Moodle pour trouver, c'est **votre responsabilité**!

Présentation

L'objectif de ce travail pratique est de mettre en application l'algorithme de compression sans perte LZ77. Il existe un nombre de variantes de cet algorithme qui reposent sur la même procédure de codage, mais se différencient par le format de représentation du code. Dans le cadre de ce TP, on s'intéressera à implémenter la version originale de l'algorithme (décrite ci-dessous), car elle est plus adaptée pour un codage en Matlab/C++, et à l'utiliser pour la compression d'images.

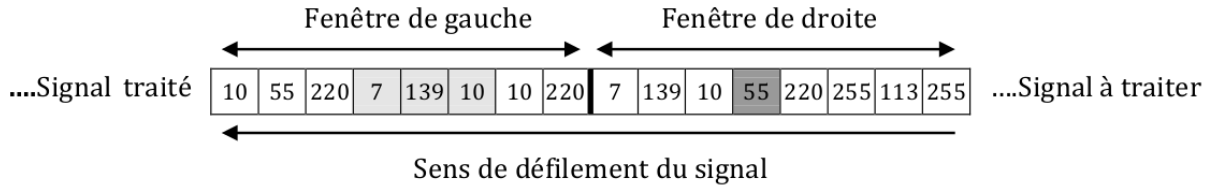
Note : les signatures des fonctions à compléter vous sont déjà fournies sur Moodle; celles-ci servent à simplifier la correction, alors ne les modifiez pas! D'ailleurs, seul le contenu de ces fonctions sera corrigé, et le code que vous utiliserez pour vos tests ne le sera pas.

L'algorithme LZ77 (Lempel et Ziv, 1977)

Tous les algorithmes Lempel-Ziv se basent sur l'étude de répétition des symboles. La version originale de LZ77 proposée par Lempel et Ziv en 1977 procède de la manière suivante :

> On fait passer le signal à coder dans une fenêtre de taille « N » divisée en deux parties de tailles n_1 et n_2. Les symboles sont ensuite défilés de droite à gauche. Si le début de la partie droite se trouve dans la partie gauche, on code la position de la sous-chaine et sa longueur.

Voici une illustration :



Le symbole courant est à droite du trait gras et on a déjà vu tous ceux qui sont à gauche. On compare le **début** de la partie droite à la partie gauche en cherchant des séquences identiques. À l'issue de la recherche de correspondance, on récupère un couple *(position, longueur),* qui peut éventuellement donner *(0,0)*. Dans l'exemple précédent, on obtient *(5,3)* : trois symboles à partir du 5^{e} (le '220' dans la fenêtre de gauche étant numéroté comme 1er symbole, le 5^{e} est donc '7'). Cela signifie que le symbole à la position n_1+*longueur+1* (i.e. le '55') ne correspond pas. On le rajoute au codage pour obtenir le code *(5,3,'55')*. La sortie de l'algorithme LZ77 est ainsi un triplet *(position, longueur, suivant)*, où :

- ***position*** : indice de 1 à n_1 de la partie de droite trouvée à gauche
- ***longueur*** : la longueur de la série de symboles trouvée à gauche
- ***suivant*** : le premier symbole de droite qui n'a pas été trouvé à gauche.

Note : pour obtenir les résultats attendus, vous devez utiliser le premier « match » identifié dans la fenêtre de gauche (en débutant votre recherche à la jonction, et en s'éloignant vers la gauche), même si celui-ci n'est pas nécessairement idéal.

Un autre exemple…

Soit le signal d'entrée : 00101021021021240101021 00, et

n_1=*9* (taille de la fenêtre de gauche);

N=18 (taille de toute la fenêtre glissante).

On initialise l'algorithme en remplissant la fenêtre de gauche (le dictionnaire) par des zéros et on charge dans la fenêtre de droite (fenêtre de lecture) les (N-n_1)=n_2=9 premiers symboles du signal. L'algorithme cherche la plus longue suite de symboles dans le dictionnaire. Puisque ce dernier contient des zéros, on pourra utiliser n'importe quelle chaine de longueur 2. Dans cet exemple, on utilise la chaine commençant à la position 1 (dernière position). Notons ici que la correspondance s'étend jusqu'à la fenêtre de lecture.

INITIALISATION

idx=9	idx=8	idx=7	idx=6	idx=5	idx=4	idx=3	idx=2	idx=1									
0	0	0	0	0	0	0	0	0	0	0	1	0	1	0	2	1	0

1. C_1 = (1,2,'1') = chaîne commence à idx=1, de longueur L=2, et se termine au symbole « 1 »

...on glisse tout vers la gauche de « L+1 » unités...

idx=9	idx=8	idx=7	idx=6	idx=5	idx=4	idx=3	idx=2	idx=1									
0	0	0	0	0	0	0	0	1	0	1	0	2	1	0	2	1	0

2. C_2=(2,3,'2') = chaîne commence à idx=2, de longueur L=3, et se termine au symbole « 2 »

0	0	0	0	1	0	1	0	2	1	0	2	1	0	2	1	2	4

3. C_3=(3,7,'2')

2	1	0	2	1	0	2	1	2	4	0	1	0	1	0	2	1	0

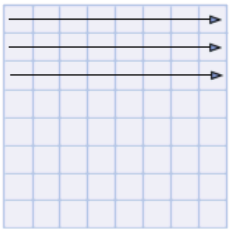
4. C_4=(0,0,'4')

1	0	2	1	0	2	1	2	4	0	1	0	1	0	2	1	0	0

5. C_5=(5,1,'1')
6. C_6=(2,3,'2')
7. C_7=(3,2,'0')

Travail demandé

A - Complétez la fonction « lz77_encode » permettant de compresser un signal tel que celui de l'exemple ci-dessus en utilisant une fenêtre glissante et un dictionnaire de taille variable. Notez ici qu'il faut utiliser le type **« unsigned integer 8 bits » (Matlab : uint8**, C++ : **uchar/uint8_t)** pour représenter les symboles du signal à coder. Chaque élément du triplet (position, longueur, suivant) sera également représenté par le type **uint8**. Par la suite, complétez aussi la fonction « **lz77_decode** » pour valider le fonctionnement de votre implémentation d'encodage (vous devriez réobtenir le signal d'origine!).



B - Complétez la fonction « format_signal » permettant de transformer une image quelconque déjà chargée en chaîne d'entiers de type « **uint8** » représentant les valeurs de tous ses pixels. Vous devez lire les pixels de l'image **ligne par ligne**, tel qu'indiqué dans la figure ci-contre. Si l'image sélectionnée est de format RGB, les composantes devraient être fournies « séquentiellement » au lieu d'être mélangées, c'est à dire qu'il faut obtenir « ...**RRRR...GGGG...BBBB... »** au lieu de « ...**RGBRGBRGBRGB... »**. La fonction inverse « **reformat_image** » vous est déjà fournie; vous pouvez vous en inspirer pour votre propre implémentation.

C - Testez votre encodage LZ77 sur toutes les images fournies avec l'énoncé en utilisant la fonction complétée à l'étape **B**, et calculez le taux de compression obtenu pour chacune à l'aide de la formule suivante :

Taux de Compr. = 1 - (Longueur du signal compressé / Longueur du signal original)

Commentez, pour chaque image, l'efficacité de l'encodage, et **expliquez** pourquoi certaines images sont mieux compressées que d'autres.

D – Dans votre rapport, proposez une amélioration à LZ77 pour augmenter l'efficacité de l'encodage sur les images à votre disposition. Souvenez-vous que normalement, LZ77 n'est jamais utilisé seul dans les 'vrais' algorithmes de compression…

Références supplémentaires

- Aide-mémoire (« Cheat sheet ») Matlab :
  - http://web.mit.edu/18.06/www/Spring09/matlab-cheatsheet.pdf
- Guide complet Matlab :
  - http://www.mathworks.com/help/pdf_doc/matlab/getstart.pdf
- Informations supplémentaires sur LZ77 et ses variations :
  - http://en.wikipedia.org/wiki/LZ77_and_LZ78

Barême

- **Implémentation et fonctionnement :**
  - lz77_encode = 6 pts
  - lz77_decode = 3 pts
  - format_signal = 2 pts
- **Rapport :**
  - Présentation de l'algo et de l'implémentation = 2 pts
  - Validité des taux de compression = 2 pts
  - Commentaires et discussion sur efficacité = 2 pts
  - Amélioration proposée (clarté et pertinence) = 1 pts
  - Lisibilité, propreté et complétude = 2 pts

(Total sur 20 pts)