
Encodeur/Décodeur d'image

Sur la base des éléments développés en TP (images différentielles) et du TD "Binaire" présentant un encodage de type VLC en pseudo-Huffman, il s'agit de développer une petite application graphique pour un CoDec (Encodeur/Décodeur) de compression d'image basé sur un codage différentiel.

Ce projet est à réaliser seul ou en binôme (donc au plus 2 personnes...)

Cahier des charges

Le point fondamental dans l'écriture d'un CoDec est le respect du Format d'Echange de Fichier (ou "FIF" en anglais, pour "File Interchange Format") : un format d'échange est une description précise de la structure d'un fichier (au bit près) qui garanti qu'une image encodée par un Encodeur soit décodable par n'importe quel Décodeur.

Cette contrainte essentielle va impliquer quelques choix, arbitraires, **qui doivent impérativement être respectés**. Ce format, que nous appellerons **DIFF**, sera décrit un peu plus loin.

Le reste du cahier des charges est beaucoup moins contraint et beaucoup de choix sont libres, en particulier en ce qui concerne les structures d'implémentations.

encodeur **pgmtodif** :

Votre application devra, au minimum :

- traiter des images en niveaux de gris (une valeur par pixel, sur 1 octet type `unsigned char`)
- lire le nom du fichier image en ligne de commande (`$> pgmtodif ./PGM/truc.pgm`)
- ouvrir un fenêtre graphique (lib. `<g2x>`) affichant l'image initiale en niveau de gris
- disposer de quelques interactions (`switch`, `button...`) permettant de :
 - visualiser l'image différentielle
 - évaluer et afficher le taux de compression atteint
 - compresser l'image au format DIFF imposé dans un fichier d'extension `./DIFF/truc.dif`
 - visualiser les histogrammes de toutes les images

décodeur **diftopgm** : en grande partie, c'est le symétrique de l'encodeur

- traiter des images au format DIFF (les vôtres ou celles de vos camarades !)
 - lire le nom du fichier image en ligne de commande (`$> diftopgm ./DIFF/truc.dif`)
 - décompresser ce fichier pour reformer une image différentielle
 - ouvrir un fenêtre graphique (lib. `<g2x>`) affichant cette image différentielle
 - disposer de quelques interactions (`switch`, `button...`) permettant de :
 - visualiser l'image niveaux de gris reconstruite (`diftpix()`)
 - enregistrer cette image au format PGM `./PGM/truc.dif.pgm`
 - visualiser les histogrammes de toutes les images
-

Architecture de code

- l'essentiel de la partie *interface graphique* a été vue en TP, de même que la partie *Image Différentielle*.
- normalement vous disposez déjà d'un module `./src/imgdif.c`, `./include/imgdif.h`
- de même, vous disposez déjà d'un `Makefile` avec une 1^o règle spécifique adaptée.

Il vous reste à écrire un module permettant la conversion de l'image différentielle (type `DifImg`) en flux binaire, selon le quantificateur VLC et la structure de fichier décrits dans les sections suivantes.

Ce module devra bien sûr être intégré au `Makefile`, dans 2 nouvelles règles spécifiques (une pour fabriquer l'encodeur `pgmtodif`, l'autre pour fabriquer le décodeur `diftojpgm`).

Choix du Quantificateur

Sur le même principe que ce qui est décrit dans le TD "Binaire", nous choisissons un Quantificateur à 4 niveaux plus ou moins arbitraire (mais testé sur un grand nombre d'images).

Le quantificateur sera défini ici par les 4 niveaux suivants :

intervalle	nbre de bits	offset	préfixe	valeur	coût
$x \in [0, 2[$	1	0	0	$x - \text{offset}$	(1 + 1) bits
$x \in [2, 6[$	2	2	10	$x - \text{offset}$	(2 + 2) bits
$x \in [6, 22[$	4	6	110	$x - \text{offset}$	(3 + 4) bits
$x \in [22, 256[$	8	22	111	$x - \text{offset}$	(3 + 8) bits


Chaque niveau est donc caractérisé par :


- un intervalle (bornes min et max)
- un nombre de bits n , déterminé par la longueur de l'intervalle
- une valeur d'*offset* (décalage – c'est sa borne min)
- un code préfixe VLC (sur p bits) de type «Huffman»

Comme dans le TD "Binaire", il suffit donc de déterminer pour chaque valeur d'entrée x , quel est son intervalle de quantification, pour coder le couple $\{\text{prfx}, (x - \text{offset})\}$ sur $(p+n)$ bits.

Adaptation : Images Différentielles

A part le choix du quantificateur, la seule différence notable avec le TD "Binaire" est qu'ici il s'agit de coder des valeurs différentielles, donc positives **ou** négatives. Le quantificateur traitera donc les **valeurs absolues** ($\in [0, 255]$) et il faudra coder **en plus** le signe (sur 1 bit – on fixera, arbitrairement, 0 pour les valeurs positives, 1 pour les négatives).

 **attention** : les données différentielles commencent au second pixel de l'image. Le premier pixel est codé à part, sous la forme d'un `uchar`, dans le champs `.first` de l'objet `DifImg`. Il sera encodé séparément dans le fichier de sortie (cf. description du fichier, vers la fin du document).

 une valeur différentielle d (valeur de «pixel» lue dans l'objet `ImgDif`) sera donc encodée sous la forme d'un triplet : $\{\text{prfx}, (|d| - \text{offset}), \text{sign}\}$ sur $(p+n+1)$ bits.

Par exemple, si les 5 premières valeurs de la source différentielle sont $(-1/0/+2/+21/-40)$, les triplets à encoder seront : $(0b0|1|1)/(0b0|0|0)/(0b10|(2-2)|0)/(0b110|(21-6)|0)/(0b111|(40-22)|1)$ – soit $3+3+5+8+12=31$ bits

et les premiers octets du buffer d'encodage seront :

`0 1 1 / 0 0 0 / 10 00 0 / 110 1111 0 / 111 00010010 1` \Rightarrow `01100010 00011011 11011100 01001010`

Encodage/Décodage

Pour tout le reste, l'encodage/décodage se déroule de manière à peu près identique à ce qui est décrit dans le TD "Binaire", avec la gestion du signe en plus.

Avant d'écrire ou de relire le fichier `./DIFF/truc.dif`, on passera par l'intermédiaire d'un *buffer* d'écriture/lecture, c'est-à-dire un gros tableau d'octets (`uchar`) que l'on remplira/videra grâce aux fonctions `pushbits/pullbits`.

☞ **attention (1)** : il y a ici plusieurs types à manipuler soigneusement

- les données «image» sont des entiers non-signés sur 1 octet : `uchar`
- les données «différentielles» sont des entiers signés sur 2 octets : `dword:signed short`
- les données à écrire dans le buffer sont des octets, donc des `uchar` \Rightarrow un `dword` est représenté par un «triplet» d'`uchar` (`prfx,|val|,signe`)
- enfin, chacun de ces 3 `uchar` est «poussé» dans le *buffer* avec juste le nombre de bits nécessaire

☞ **attention (2)** : ici on n'a aucune garantie que le codage soit effectivement compressif. Le fichier en sortie peut être plus gros que celui d'entrée.

Pour évaluer la taille que doit avoir ce *buffer*, il faut envisager le pire des cas : celui où toutes les valeurs différentielles sont dans le dernier intervalle (`[22,255]` pour notre quantificateur) : chaque valeur différentielle coûte alors $(3+8+1)=12$ bits soit 1,5 fois «plus cher» qu'un codage direct.

☞ il suffit donc, connaissant la taille de l'image ($N=\text{width}*\text{height}$), d'allouer un *buffer* de taille $(1,5*N)$ octets

Format d'Echange de Fichier - Spécifications

On en arrive à la partie la plus importante : la description du format du fichier d'encodage (le format `DIFF`) qui doit être respecté au bit près.

Ⓐ **l'en-tête** : contient les données indispensables à l'identification du format et à son décodage

- le «Magic Number» (ou code d'identification) : le format `DIFF` sera identifié par les deux premiers octets (`unsigned short`) du fichier qui devront être `0xD1FF`
- les tailles de l'image : largeur (`width`) et hauteur (`height`), chacune sous la forme de deux `unsigned short` (2 octets chacun)
- les informations du Quantificateur : il suffit d'indiquer
 - le nombre de niveaux, sur 1 octet (ici c'est toujours 4, mais il faut le préciser)
 - le nombre de bits n associé à chaque niveau. Donc ici, 4 octets contenant les valeurs 1,2,4,8☞ le décodeur peut ainsi *reconstruire* les intervalles `[0,2[`, `[2,6[`, `[6,22[`, `[22,256[`, les valeurs d'offset et les préfixes (code de Huffman optimal `0b0|0b10|0b110|0b111`)

L'en-tête coûte donc toujours 11 octets et seules les tailles changent d'une image à l'autre.

Ⓒ **le premier octet** : c'est la valeur du champ `.first` de l'objet `DifImg`, c'est à dire la valeur brute du premier pixel de l'image.

Ⓓ **les données compressées** : c'est tout simplement le *buffer* rempli à l'étape précédente.

Exemple :

Pour une image d'entrée de tailles 620x440 commençant par les valeurs de pixel `90|90|91|90|88|99|99|94|...`, le fichier `.dif` en sortie commencera comme suit :

magic (2)	size (2x2)	quantif.(1+4x1)	.first (1)	buffer : compressed data
0xD1FF	0x026C 0x01B8	0x04 0x01 0x02 0x04 0x08	0x5A	00001001 11000111 00010100 0010111...

☞ Toutes les données à écrire/lire étant «brutes» (binaires) il faut utiliser les fonctions `fwrite` et `fread` de la libC. Le *buffer* peut être traité en bloc, en une seule fois.

Décodeur :

La mise en œuvre du décodeur n'est pas plus compliquée : c'est la même chose, à l'envers.

Ⓐ lecture de l'en-tête

- identification du «magic number» (message d'erreur et arrêt si ce n'est pas le bon)
- lecture des tailles $N = \text{width} * \text{height}$ → création (alloc.) de l'image différentielle (ImgDif)
- lecture des paramètres du quantificateur et reconstruction (`préfixe,min,max,nbit,offset`)

Ⓑ décompression

- lecture de la première valeur → champs `.first` de l'objet `ImgDif`
 - chargement du buffer (taille par défaut $1,5 * N$)
 - lecture (`pullbits`) des triplets (`prfx,|val|,sign`)
 - `préfixe` (de 1 à 3 bits) → intervalle du quantif. → `{nbit,offset}`
 - `|val|` (nbit bits) et du signe `s` (1 bit) → valeur différentielle $d = s * (|val| + \text{offset})$ → `ImgDif`.
- arrêt lorsque les $(N-1)$ valeurs différentielles ont été extraites et reconstruites.

Ⓒ reconstruction (`diftoPix`) de l'image en niveaux de gris

Ⓓ interface graphique : affichages, histogrammes, enregistrement (`.pgm`)

Rendu du projet :

Votre projet devra être déposé sur eLearning au plus tard le [vendredi 17 Janvier 2025, 23h59](#) sous la forme d'une archive zip de nom `B13.Nom1.Nom2.zip` (Nom1, Nom2 : les noms du binôme, bien sûr...).

Elle doit se dézipper sous la forme d'un répertoire `B13.Nom1.Nom2/` propre contenant :

- un sous-répertoire `src/` contenant les sources (`*.c`)
- un sous-répertoire `include/` contenant les headers (`*.h`)
- le `Makefile`, avec les bonnes règles de compilation (codeur/décodeur)
- un fichier texte brut (aucun format) `README` contenant une description sommaire de votre projet :
 - Nom/Prénom des participants
 - appel de compilation
 - exemples d'appels d'exécution (options de la ligne de commande)
 - répartition du travail entre les participants (qui a fait plus ou moins quoi)
 - une description de ce qui marche ou ne marche pas, des difficultés rencontrées, des améliorations/options apportées...
- votre archive **ne doit pas** contenir d'image (`.pgm` ou `.dif`) ni de `.pdf`, fichier objet (`.o`) ou exécutable
→ uniquement des fichiers texte (codes sources, `Makefile`, `README`)