Algorithmique des images TD n° 1bis

Manupulation simple d'images

Exercice 1. Manipulation simple d'images "Gray scale" (PGM)

Le format de fichier **Portable GrayMap** (pgm) permet d'enregistrer des images en niveaux de gris ou chaque pixel est codé par une valeur entière. Ainsi en ouvrant une image de format (pgm) préalablement enregistrée en ASCII avec un éditeur de texte type gedit (ou bien emacs) vous pourrez observer le code de chaque pixel. Par exemple en ouvrant le fichier eye_s_asc.pgm vous observerez le format suivant :

```
1   P2         <-- Code pour le format PGM
2   # Created by GIMP version 2.10.36 PNM plug-in
3   512 423    <-- Largeur et hauteur de l'image
4   255        <-- pixel 0
5   144        <-- pixel 1
6   144        <-- pixel 2
7   143        <-- pixel 3
8   145        <-- pixel 4
9   145        <-- pixel 5
10  144        <-- pixel 6
11  ...
```

Q- 1.1 Structure

Ecrire une structure pgm qui contiendra la hauteur (height), la largeur (width) la valeur maximale de codage des pixels (max_value) ainsi qu'un pointeur sur un tableau à deux dimensions de caractères non-signés (pixels).

Q- 1.2 Allocation

Écrire une fonction pgm_alloc qui prend en paramètre la hauteur (height), la largeur (width) la valeur maximale de codage des pixels (max_value) et qui retourne une structure pgm contenant les données d'une image de taille height x width de pixels initialisés à la valeur max_value.

Q- 1.3 Libération

Écrire une fonction pgm_free qui prend en paramètre un pointeur sur une structure pgm et libère l'espace mémoire occupé par cette structure.

Q- 1.4 Lecture ASCII

Écrire la fonction pgm_read_asc qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier au format ASCII à lire (fname) et retournant un pointeur sur une structure pgm contenant les informations relatives à l'image contenue dans le fichier fname.

Q- 1.5 Écriture ASCII

Écrire la fonction pgm_write_asc qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier (fname) à écrire (au format ASCII) ainsi qu'un pointeur sur une structure pgm. La fonction retournera un entier égale à 0 si tout s'est bien passé et à 1 sinon.

Q- 1.6 Lecture BINAIRE

Écrire la fonction `pgm_read_bin` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier au format `BINAIRE` à lire (`fname`) et retournant un pointeur sur une structure `pgm` contenant les informations relatives à l'image contenue dans le fichier `fname`.

Q- 1.7 Écriture BINAIRE

Écrire la fonction `pgm_write_bin` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier (`fname` à écrire (au format `BINAIRE`) ainsi qu'un pointeur sur une structure `pgm`. La fonction retournera un entier égale à 0 si tout s'est bien passé et à 1 sinon.

Q- 1.8 Négatif

Écrire la fonction `pgm_negative` qui prendra en paramètre un pointeur `scr` sur une structure `pgm` contenant l'image source et un pointeur `dst` sur une structure `pgm` contenant le négatif de l'image source.

Q- 1.9 Extraction

Écrire la fonction `pgm_extract` qui en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier de sortie (`fname`), une structure `pgm_t`, les coordonnées `dx` et `dy` indiquant le point de départ de l'image à extraire et les dimensions de l'image à extraire `width` et `height`. La fonction écrira dans le fichier `fname` une "sous-image" extraite de l'image principale.

Q- 1.10 Histogramme

Écrire la fonction `pgm_get_histrogram` qui prendra en paramètre un pointeur sur une structure `pgm` et qui retournera un pointeur sur un tableau de `max_value` contenant l'histogramme des pixels de l'image.

Q- 1.11 Fichier histogramme

Écrire la fonction `pgm_write_histogram` qui prendra en paramètre un pointeur sur une structure `pgm`, un pointeur sur une chaine de caractère `fname`. La fonction devra créer le fichier `fname` et l'histogramme de l'image sous la forme de deux colonnes (la première colonne contiendra les valeurs de 0 à `max_value`, la seconde les données de l'histogramme correspondant).

Q- 1.12 Test

Écrire un programme de test `test_pgm.c` afin de tester toutes les fonctions précédentes.

Exercice 2. Manipulation simple d'images "Couleurs" (PPM)

Le format de fichier **Portable PixMap** (`ppm`) permet d'enregistrer des images en couleur ou chaque pixel est codé en RGB. Le système RGB code un pixel par 3 octets, chaque octet prend donc une valeur entre 0 et 255, donnant respectivement le ton de rouge, vert et bleu. Ainsi en ouvrant une image de format (`ppm`) préalablement enregistrée en `ASCII` avec un éditeur de texte type `gedit` (ou bien `emacs`) vous pourrez observer le code RGB de chaque pixel. Par exemple en ouvrant le fichier `eye_s_asc.ppm` vous observerez le format suivant :

```
1  P3          <-- Code pour le format PPM
2  # Created by GIMP version 2.10.36 PNM plug-in
3  512 423     <-- largeur et hauteur
4  255         <-- Valeur Max d'une composante couleur
5  181         <-- code R du pixel 0
```

```
    133         <-- code G du pixel 0
    110         <-- code B du pixel 1
    181         <-- code R du pixel 1
    133         <-- code G du pixel 1
    ...
```

où les pixels sont numérotés ligne par ligne. Le pixel du coin supérieur gauche est à la position 0, son voisin de droite est à la position 1 et ainsi de suite :

0	1	2	3	4
5	6	7	8	9
10	11	12	13	14

```c
struct rgb
{
  unsigned char r, g, b;
};
typedef struct rgb rgb_t;
```

Pour simplifier, nous supposerons que les valeurs des composantes RBG sont limitées à 255.

Q- 2.1 Structure

Écrire une structure `ppm` qui contiendra la hauteur (`height`), la largeur (`width`), une valeur maximale (`max_value`) et un pointeur sur un tableau à deux dimensions de structure `rgb` (`pixels`).

Q- 2.2 Allocation

Écrire une fonction `ppm_alloc` qui prend en paramètre la hauteur (`height`), la largeur (`width`) la valeur maximale de codage des pixels (`max_value`) et qui retourne une structure `ppm` contenant les données d'une image de taille `height` x `width` de pixels initialisés à la valeur `max_value`.

Q- 2.3 Libération

Écrire une fonction `ppm_free` qui prend en paramètre un pointeur sur une structure `ppm` et libère l'espace mémoire occupé par cette structure.

Q- 2.4 Lecture ASCII

Écrire la fonction `ppm_read_asc` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier au format `ASCII` à lire (`fname`) et retournant un pointeur sur une structure `ppm` contenant les informations relatives à l'image contenue dans le fichier `fname`.

Q- 2.5 Écriture ASCII

Écrire la fonction `ppm_write_asc` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier (`fname`) à écrire (au format `ASCII`) ainsi qu'un pointeur sur une structure `ppm`. La fonction retournera un entier égale à 0 si tout s'est bien passé et à 1 sinon.

Q- 2.6 Lecture BINAIRE

Écrire la fonction `ppm_read_bin` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier au format `BINAIRE` à lire (`fname`) et retournant un pointeur sur une structure `ppm` contenant les informations relatives à l'image contenue dans le fichier `fname`.

Q- 2.7 Écriture BINAIRE

Écrire la fonction `ppm_write_bin` qui prendra en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier (`fname` à écrire (au format `BINAIRE`) ainsi qu'un pointeur sur une structure `ppm`. La fonction retournera un entier égale à 0 si tout s'est bien passé et à 1 sinon.

Q- 2.8 Négatif

Écrire la fonction `ppm_negative` qui prendra en paramètre un pointeur `scr` sur une structure `ppm` contenant l'image source et un pointeur `dst` sur une structure `ppm` contenant le négatif de l'image source.

Q- 2.9 Extraction

Écrire la fonction `ppm_extract` qui en paramètre un pointeur sur une chaine de caractères contenant le nom du fichier de sortie (`fname`), une structure `ppm_t`, les coordonnées `dx` et `dy` indiquant le point de départ de l'image à extraire et les dimensions de l'image à extraire `width` et `height`. La fonction écrira dans le fichier `fname` une "sous-image" extraite de l'image principale.

Q- 2.10 Histogramme

Écrire la fonction `ppm_get_histrogram` qui prendra en paramètre un pointeur sur une structure `ppm` et qui retournera un pointeur sur un tableu à deux dimensions (3,=$\max_{\text{value}}$=) contenant les histogrammes des trois composantes `RGB` des pixels de l'image.

Q- 2.11 Fichier histogramme

Écrire la fonction `ppm_write_histogram` qui prendra en paramètre un pointeur sur une structure `ppm`, un pointeur sur une chaine de caractère `fname`. La fonction devra créer le fichier `fname` et l'histogramme de l'image sous la forme de quatre colonnes (la première colonne contiendra les valeurs de 0 à `max_value`, les trois colonnes suivantes les données de l'histogramme des trois composantes correspondant).

Q- 2.12 Conversion

Écrire la fonction `ppm_to_pgm` qui prendra en paramètre un pointeur sur une structure `ppm` et un pointeur sur une structure `pgm`. La fonction convertira l'image `ppm` en image `pgm`.

Q- 2.13 Test

Écrire un programme de test `test_ppm.c` afin de tester toutes les fonctions précédentes.

Exercice 3. Compression JPEG

De manière générale la compression JPEG permet de compresser des images couleurs. Par souci de simplicité, nous nous restreindrons ici à la compression de la composante Y d'une telle image, c'est à dire d'une image au format pgm. La première étape de la compression consiste à subdiviser l'image en bloc de 8×8 pixels et d'appliquer une transformée en cosinus discrète à chaque bloc. On rappelle ci dessous la formule de la transformée en cosinus discrète en deux dimensions pour un bloc de taille N × N :

$$\texttt{DCT}(i,j) = \frac{2}{N} C(i) C(j) \sum_{x=0}^{N-1} \sum_{y=0}^{N-1} \texttt{pixel}(x,y) \cos\left(\frac{(2x+1)i\pi}{2N}\right) \cos\left(\frac{(2y+1)j\pi}{2N}\right),$$

avec $C(z) = 1$ pour $z > 0$ et $C(0) = 1/\sqrt{2}$. Une fois la transformée appliquée, on applique l'étape de quantification qui consiste à diviser le coefficient `DCT`(i, j) d'un bloc par le coefficient $Q_{i,j}$ de la matrice de quantification Q donnée ci dessous :

$$Q = \begin{pmatrix} 16 & 11 & 10 & 16 & 24 & 40 & 51 & 61 \\ 12 & 12 & 14 & 19 & 26 & 58 & 60 & 55 \\ 14 & 13 & 16 & 24 & 40 & 57 & 69 & 56 \\ 14 & 17 & 22 & 29 & 51 & 87 & 80 & 62 \\ 18 & 22 & 37 & 56 & 68 & 109 & 103 & 77 \\ 24 & 35 & 55 & 64 & 81 & 104 & 113 & 92 \\ 49 & 64 & 78 & 87 & 103 & 121 & 120 & 101 \\ 72 & 92 & 95 & 98 & 112 & 100 & 103 & 99 \end{pmatrix}.$$

Le résultat de la division sera arrondi à l'entier le plus proche. À partir de ce moment là, les seules valeurs non nulles seront présentes dans la partie supérieure gauche du bloc 8×8. Afin de pouvoir réaliser une compression RLE du résultat, on extrait les valeurs du bloc en zigzag comme indiqué dans la figure 1.

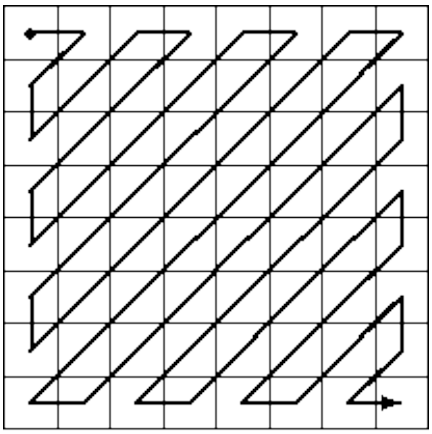

Figure 1: Extraction en zigzag des valeurs d'un bloc 8×8

de sorte à obtenir une liste de 64 entiers dont les dernières valeurs sont toutes nulles. Ces dernières valeurs seront compressées à l'aide d'un algorithme RLE. Dans cette partie on supposera que les images passées en paramètres ont des dimensions (largeur et hauteur) divisibles par 8.

Q- 3.1 Extraction par bloc

Créer une fonction `void pgm_extract_blk(pgm_t *inpgm, double blk[8][8], int i, int j)` qui extrait le bloc 8×8 formé de la composante Y de l'image `ppm` pointée par `image` dont le coin supérieur gauche se trouve aux coordonnées (i, j). Ce bloc sera sauvegardé dans le tableau de double `bloc` passé en paramètre.

Q- 3.2 DCT

Créer une fonction `void pgm_dct(double bloc[8][8])` qui applique la transformée en cosinus discrète bi-dimensionnelle à un tableau `bloc` de taille 8×8. Pensez à vérifier le bon fonctionnement de votre fonction avec l'exemple du cours.

Q- 3.3 Quantification

Créer la fonction `void pgm_quantify(double blk[8][8], double Q[8][8])` qui quantifie le bloc `blk` passé en paramètre avec la matrice de quantification Q passée en paramètre.

Q- 3.4 Parcours en zig-zag

Créer la fonction `void pgm_zigzag_extract(double blk[8][8], int zgzg[64])` qui extrait les 64 nombres contenus dans le bloc `blk` de taille 8×8 dans l'ordre donné par la figure 1. Les valeurs de `blk` seront arrondies à l'entier le plus proche avant d'être stockées dans le tableau `zgzg` passé en paramètre.

Q- 3.5 Compression RLE

Créez la fonction `void pgm_rle(FILE *fd, int zgzg[64])` qui écrit les entiers contenus dans le tableau `zgzg` dans le fichier pointé par `fd`. On supposera que le flux donné par `fd` aura été ouvert préalablement. Chaque entier sera écrit sur une ligne différente et une séquence de n 0 sera codée par `@$n$` dès que $n \geq 2$.

Q- 3.6 Compression JPEG

À l'aide des fonctions précédentes, créer une fonction `void pgm_to_jpeg(pgm_t *in_pgm, char *fname)` qui compresse l'image `pgm` pointée par `in_pgm` en utilisant l'algorithme de compression JPEG et qui stocke le résultat dans un fichier dont le nom est donné par `fname`. Le fichier compressé respectera le format suivant :

```
JPEG
width height
valeur 0
valeur 1
valeur 2
...
...
```

où `width` et `height` correspondront à la largeur et la hauteur de l'image compressée.

Q- 3.7 Taille de l'image

Créez une fonction `unsigned int fsize(char *fname)` qui renvoit la taille en octets du fichier nommé par la chaine de caratère `fname`. On supposera que chaque caractère du fichier `fname` est codé sur 1 octet. Vous pourrez vérifiez le résultat de votre fonction en le comparant avec le résultat de la commande `wc`.

Q- 3.8 Tests

Créer un programme de test pour compresser une image `pgm`. Vous pourez utilisé une image extraite de l'image `eye_s_asc.pgm` de taille 256 par 256.

Votre programme affichera également la taille du fichier `pgm`, la taille du fichier `jpeg` ainsi que le taux de compression.

Exercice 4. Décompression JPEG (Exercices Bonus)

Q- 4.1 Fonctions

Réaliser toutes les fonctions inverses du processus de compression.

Q- 4.2 Décompression

Réasiler la fonction `pgm_t *jpeg_to_pgm(char *fname)` qui lit les données d'une image compressée au format `jpeg` et qui retourne un pointeur sur une structure `pgm_t` contenant l'image.

Q- 4.3 Tests

Ecrire un programme de tests.

Exercice 5. Compression JPEG des images couleurs (Exercices Bonus)

Q- 5.1 Fonctions

Reprendre toutes les fonctions de compression des images `pgm` pour les appliquer aux images `ppm`. Ne pas oublier de rajouter les fonctions nécessaires au passage de `RBG` à `YUV`.

Q- 5.2 Tests

Résaliser un programme de tests permettant de compresser des images au format `ppm`.

Exercice 6. Décompression JPEG des images couleurs (Exercices Bonus)

Q- 6.1 Fonctions

Réaliser toutes les fonctions inverses du processus de compression des images couleurs.

Q- 6.2 Décompression

Réasiler la fonction `ppm_t *jpeg_to_ppm(char *fname)` qui lit les données d'une image compressée au format `jpeg` et qui retourne un pointeur sur une structure `ppm_t` contenant l'image.

Q- 6.3 Tests

Ecrire un programme de tests permettant de décompresser des images au format `jpeg`.