Programmation avancée

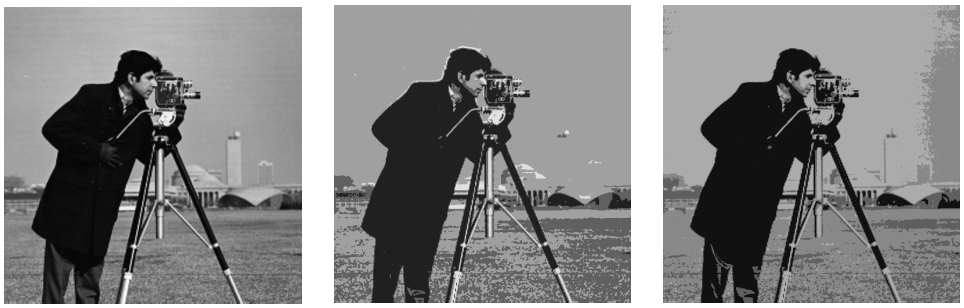
Projet 3: analyse d'images

Jean-Michel Begon – Romain Mormont – Pascal Fontaine

27 novembre 2020

COVID-19. Au vu de la situation sanitaire actuelle et des difficultés que celle-ci engendre, *tout* ce qui concerne la variante gloutonne (questions 3 et partiellement 7 et 8, ainsi que l'implémentation) est optionnel et sera considéré comme bonus si vous le faites.

Ce projet vise à illustrer les techniques de résolution de problèmes pour mettre au point un algorithme de compression d'images. L'objectif est d'écrire une fonction permettant de faire passer de n à k, avec $k < n$, le nombre de niveaux de gris nécessaires à l'encodage d'une image en préservant autant que possible la qualité de l'image de départ (figure 1).



(a) 256 niveaux de gris. (b) Compression naïve (c) Compression optimale

Figure 1 – Compressions d'une image de 256 vers 4 niveaux.

1 Formalisation du problème

Soit l'image de départ représentée par une matrice I de taille $w \times h$ telle que $I[i,j] \in \{0,\dots,n-1\}$ est le niveau du pixel (i,j). Typiquement, 0 correspond au noir, 255 représente le blanc et les valeurs intermédiaires correspondent aux nuances de gris.

La compression que nous étudions ici consiste à associer à chaque niveau $v \in \{0,\dots,n-1\}$ une nouvelle valeur parmi k valeurs $\{v_1, v_2, \dots, v_k\} \subseteq \{0,\dots,n-1\}$ telle que définie par une fonction de compression $g : \{0,\dots,n-1\} \to \{v_1, v_2, \dots, v_k\}$. On définira l'*erreur* $Err(g)$ associée à la compression définie par la fonction g comme la somme sur tous les pixels de l'image du carré de la différence entre la valeur originale et la valeur compressée :

$$Err(g) = \sum_{i=1}^{w} \sum_{j=1}^{h} (I[i,j] - g(I[i,j]))^2 \tag{1}$$

Cette somme peut être réécrite sous la forme d'une somme sur toutes les valeurs de pixels :

$$Err(g) = \sum_{i=0}^{n-1} h[i](i - g(i))^2, \tag{2}$$

où $h[i]$, pour $i = 0, \ldots, n-1$, est le nombre de pixels de l'image de valeur i (c'est-à-dire l'*histogramme* de l'image).

L'objectif est de déterminer la fonction g et les valeurs v_k qui minimisent cette erreur.

On peut montrer que le problème est équivalent à déterminer une partition de $\{0, \ldots, n-1\}$ en k sous-intervalles de valeurs consécutives représentées chacune par une même valeur, c'est-à-dire considérer les fonctions g de la forme :

$$g(i) = \begin{cases} v_1 \text{ si } 0 \leq i < p_1 \\ v_2 \text{ si } p_1 \leq i < p_2 \\ \ldots \\ v_k \text{ si } p_{k-1} \leq i < n \end{cases}, \tag{3}$$

où $0 = p_0 < p_1 < p_2 < \ldots < p_{k-1} < n$ et $v_1 < v_2 < \ldots < v_k$. Trouver une meilleure réduction revient donc à déterminer les $k-1$ seuils p_j, $j = 1, \ldots, k-1$ et les k valeurs v_j, $j = 1, \ldots, k$ de manière à minimiser l'erreur (2).

Exemple : Soit I une image 5×5. Notons $\hat{I}$ une compression optimale en 3 niveaux. Par exemple, nous avons le couple

$$I = \begin{pmatrix} 3 & 6 & 0 & 4 & 0 \\ 7 & 4 & 3 & 3 & 4 \\ 4 & 1 & 1 & 0 & 0 \\ 4 & 1 & 7 & 5 & 6 \\ 6 & 7 & 5 & 0 & 6 \end{pmatrix} \qquad \hat{I} = \begin{pmatrix} 4 & 6 & 0 & 4 & 0 \\ 6 & 4 & 4 & 4 & 4 \\ 4 & 0 & 0 & 0 & 0 \\ 4 & 0 & 6 & 6 & 6 \\ 6 & 6 & 6 & 0 & 6 \end{pmatrix}.$$

En effet, les 25 valeurs de I, réparties en 8 niveaux, forment l'histogramme

$$h = [5, 3, 0, 3, 5, 2, 4, 3].$$

L'erreur minimale de compression en 3 niveaux est de 11 et donne la compression optimale, correspondant à $\hat{I}$, suivante :

$$g(i) = \begin{cases} 0 \text{ si } 0 \leq i < 2 \\ 4 \text{ si } 2 \leq i < 5 \\ 6 \text{ si } 5 \leq i \leq 8 \end{cases}.$$

Enfin, remarquons qu'un critère équivalent (*i.e.* qui fournit le même minimum) mais plus facile à interpréter est

$$RMSE(g) = \sqrt{\frac{1}{w\,h} Err(g)} \tag{4}$$

où $RMSE$ signifie *root mean square error*. Par soucis de facilité, nous utiliserons cependant Err dans le code.

2 Analyse théorique

Avant de passer à l'implémentation, on se propose d'analyser trois solutions algorithmiques à ce problème : une approche exhaustive, une approche gloutonne et une approche par programmation dynamique. A l'exception de la dernière question, on supposera qu'on dispose directement du tableau h contenant l'histogramme des niveaux de gris dans l'image à compresser.

Questions

1. Déduisez que le problème se résume à trouver les p_i.
   (a) Montrez qu'il est possible de déterminer les valeurs des v_i minimisant l'erreur (2) de manière analytique une fois les p_i connus.
   (b) Discutez la complexité du calcul des v_i.
2. Expliquez le principe d'une approche par recherche exhaustive pour résoudre le problème et exprimez sa complexité en fonction de k et n.
3. Une approche gloutonne à ce problème consisterait à déterminer itérativement à chaque étape i ($i = 1, \ldots, k-1$) un nouveau p_{k_i} sur base des $\{p_{k_1}, \ldots, p_{k_{i-1}}\}$ précédemment calculés, sans remettre en question les choix précédents.
   (a) Sur base de l'histogramme de l'image, proposez et expliquez une solution selon le formalisme de votre choix suivant ce principe.
   (b) Etudiez sa complexité en fonction de k et de n.
   (c) Montrez par un contre-exemple que cette approche ne fournit pas la solution optimale dans tous les cas.

   Note : L'approche gloutonne n'a d'intérêt que si (i) la solution fournie est proche de l'optimale, et (ii) elle est plus efficace en temps que l'approche par programmation dynamique. Par exemple, tirer au hasard un seuil p_j et partitionner l'intervalle $[p_j, p_{j+1}[$ en deux en ajoutant un seuil en $\frac{p_j+p_{j+1}}{2}$ satisfait à la définition d'une approche gloutonne et est très efficace mais fournira probablement des erreurs très mauvaises par rapport à l'approche optimale.
   Il est possible d'atteindre une complexité de $\Theta(nk)$ en satisfaisant à ces deux critères, ainsi qu'à la définition de l'approche gloutonne.
4. On se propose maintenant de résoudre le problème de manière optimale en utilisant la programmation dynamique. La fonction de coût qu'on cherchera à optimiser sera l'erreur minimale, notée $ErrMin(n, k)$, obtenue en compressant de manière optimale l'histogramme entre les niveaux 0 à $n-1$ sur k niveaux.
   (a) Donnez la formulation récursive complète de la fonction de coût $ErrMin(n, k)$ en précisant bien les cas de base.
   Suggestion : inspirez vous du problème de la tige d'acier, ainsi que de votre raisonnement du point 1.
   (b) Donnez le pseudo-code d'un algorithme *efficace* de calcul des valeurs de cette fonction (par l'approche ascendante ou descendante).
   (c) Donnez la complexité en *temps* et en *espace* de cet algorithme en fonction de n et k. Est-ce que la complexité change lorsqu'on récupère aussi la solution optimale ?

Note : Pour obtenir le maximum de points à cette question, on vous demande de fournir une solution $\Theta(n^3)+\Theta(n^2k)$ au pire cas. Un bonus sera accordé pour une implémentation optimale en temps de $\Theta(n^2k)$ au pire cas.

5. Quelle est la complexité en temps au pire et au meilleur cas de la totalité de l'algorithme optimal (calcul de l'histogramme, calcul de la compression optimale, application de la compression) ?

3 Implémentation

On vous demande d'implémenter la fonction `computeMapping` définie dans le header `compression.h`. La variante gloutonne devra être implémentée dans un fichier `greedy_compression.c` alors que la variante par programmation dynamique devra être implémentée dans un fichier `dp_compression.c`.

Une méthode naïve de réduction du nombre de couleurs, qui ne tient pas compte de l'erreur, consiste à diviser l'intervalle $[0, n-1]$ en k sous-intervalles de taille approximativement égale et à associer à chaque sous-intervalle la valeur centrale de l'intervalle. Cette variante est implémentée dans le fichier `naive_compression.c` et vous est fournie pour comparaison. Les autres fichiers fournis sont décrits à la section 3.

Questions

Dans votre rapport :

6. Si nécessaire par rapport aux descriptions données aux questions 3 et 4 ci-dessus, expliquez brièvement vos implémentations de ces fonctions.
7. Vérifiez empiriquement les complexités théoriques (en temps) dérivées aux points 3b et 4c Faites pour cela des essais avec des histogrammes de tailles croissantes générés aléatoirement.
   Note : le fichier `emp-time.c` vous est fourni à cette fin. Il contient une fonction qui génère des histogrammes et une qui permet de mesurer les temps d'exécution.
8. Comparez visuellement et quantitativement vos deux approches avec l'approche naïve sur les images fournies. Discutez les résultats pour différentes valeurs de k. Concluez sur l'intérêt de ces différentes solutions.

Fichiers fournis

Afin de vous aider, nous vous fournissons également les fichiers suivants :

`main.c` le programme qui implémente la totalité de l'algorithme (chargement de l'image d'entrée, calcul de la compression "optimale", application de la compression et écriture de la nouvelle image).

`PGM.h/c` une petite libraire de gestion des images en niveaux de gris au format `pgm`.

`Mapping.h/c` les structures représentant l'histogramme, la fonction de compression, ainsi que les fonctions qui vont avec.

`*.pgm` quelques images de tests. Il est possible de générer des images dans ce format à l'aide de tous les outils standard de traitement d'image.

3.1 Utilisation

Vous pouvez compiler le `main.c` à l'aide de la commande

```
gcc main.c naive_compression.c PGM.c Mapping.c --std=c99 --pedantic -Wall
      -Wextra -Wmissing-prototypes -DNDEBUG -lm -o compress
```

Le fichier généré prend trois paramètres de la ligne de commande : le chemin vers l'image (au format `pgm`) à compresser, le nombre de niveaux après compression et le chemin vers l'image de sortie. Par exemple,

```
./compress camera.pgm 4 camera4.pgm
```

compresse `camera.pgm` (qui se trouve dans le même dossier que le programme) en 4 niveaux et sauvegarde le résultat dans `camera4.pgm` (toujours dans le même dossier).

Le choix de l'algorithme se fait à la compilation en substituant adéquatement le fichier implémentant `compression.h`.

4 Deadline et soumission

Le projet est à réaliser **binôme** pour le **20 décembre 2020** à **23h59** au plus tard. Le projet est à remettre via la plateforme `https://submit.montefiore.ulg.ac.be/`.

Il doit être rendu sous la forme d'une archive `tar.gz` contenant :

1. votre rapport au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions ;
2. le fichier `dp_compression.c` ;
3. le fichier `greedy_compression.c`.

Comme pour les autres projets

— les noms des fichiers doivent être respectés ;

— le projet doit être réalisé dans le standard C99 ;

— la présence de *warnings* impactera négativement la cote finale ;

— un projet qui ne compile pas sur les machines ms8xx recevra une cote nulle (pour la partie code du projet).

Un projet non rendu à temps recevra une cote globale nulle. En cas de plagiat avéré, l'étudiant se verra affecter une cote nulle à l'ensemble des projets.

Les critères de correction sont précisés sur la page web des projets.

Bon travail !