Structures de données et algorithmes

Projet 3: analyse d'images

Pierre Geurts – Jean-Michel Begon – Romain Mormont

24 avril 2015

Ce projet vise à illustrer les techniques de résolution de problème 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. Cette réduction entraînant immanquablement une perte de détails dans l'image, l'objectif est de l'effectuer en préservant autant que possible la qualité de l'image de départ. Un tel algorithme a beaucoup d'applications telles que par exemple la compression d'image, l'affichage d'une image sur un écran au nombre de couleurs limité ou encore comme prétraitement pour permettre d'autres opérations (identification de zones, etc.).

Formalisation du problème

Soit l'image de départ représentée par une matrice I de taille $w \times h$ tel que $I[i,j] \in \{0,\dots,n-1\}$ est le niveau du pixel (i,j). Le problème 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 $g(v)$. On définira l'*erreur* associée à la réduction comme la somme sur tous les pixels de l'image du carré de la différence entre la valeur originale et la valeur compressée :

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

où $g(.)$ est la fonction qui associe à chaque niveau sa valeur "compressée".

Cette somme peut se réécrire en une somme sur tous les niveaux :

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

où $h[i]$, pour $i = 0,\dots,n-1$, 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 minimise cette erreur.

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

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

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

Exemple : Soit l'image 5×5

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

en 8 valeurs de gris. Le vecteur du nombre de pixels par niveau est donné par :

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

Une réduction optimale en 3 niveaux est donnée par :

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

et son erreur est 11. L'image compressée est donnée par :

$$\hat{I} = \begin{pmatrix} 4 & 6 & 0 & 4 & 0 \\ 6 & 4 & 4 & 4 & 4 \\ 4 & 0 & 0 & 0 & 0 \\ 4 & 0 & 6 & 6 & 6 \\ 6 & 6 & 6 & 0 & 6 \end{pmatrix}.$$

1 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. Seules les deux dernières devront être implémentées. On supposera pour cette analyse qu'on dispose directement du tableau h contenant l'histogramme des niveaux de gris dans l'image à compresser.

Questions

1. Montrez que si les p_i sont connus, il est possible de déterminer les valeurs des v_i minimisant l'erreur (1) de manière analytique. Déduisez en que le problème algorithmique revient alors à déterminer uniquement les valeurs des p_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 cause les choix précédents.
   (a) Imaginez et expliquez selon le formalisme de votre choix une approche gloutonne 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 : Pour obtenir le maximum de points à cette question, votre approche doit fournir une solution aussi proche que possible de l'optimale et être plus efficace en temps que l'approche par programmation dynamique. Par exemple, tirer au hasard chacun des seuils p_{k_i} 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.

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 de la compression optimale de 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.
   (a) Représentez le graphe des appels récursifs pour un problème de petite taille.
   (b) Donnez le pseudo-code d'un algorithme *efficace* de calcul des valeurs de cette fonction (par l'approche ascendante ou descendante).
   (c) Modifier ce pseudo-code pour récupérer la solution optimale.
   (d) Donnez la complexité en *temps* et en *espace* de cet algorithme.

   Note : Pour obtenir le maximum de points à cette question, on vous demande de fournir la solution la plus efficace possible en temps et en espace.

2 Implémentation

On vous demande d'implémenter les fonctions définies par les interfaces `Reduction.h` et `ImageQuantizer.h` :

`computeReduction` qui calcule les valeurs de $p_1, \ldots, p_{k-1}$ et $v_1, \ldots, v_k$ à partir d'un vecteur de comptage h et une valeur de k.

`quantizeGrayImage` qui réduit le nombre de niveaux de gris d'une image à un nombre k.

L'implémentation de `computeReduction` doit se faire dans un fichier `GreedyReduction.c` pour la variante gloutonne et dans un fichier `DPReduction.c` pour la variante par programmation dynamique. La fonction `quantizeGrayImage`, commune aux deux implémentations, doit se faire dans un fichier `ImageQuantizer.c`.

Une méthode de réduction du nombre de couleurs naïve 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 `NaiveImageQuantizer.c` et vous est fournie pour comparaison.

Questions

Dans votre rapport :

5. Si nécessaire par rapport aux descriptions données aux points 3 et 4 ci-dessus, expliquez brièvement vos implémentations de ces fonctions.
6. Verifiez empiriquement les complexités théoriques (en temps) dérivées aux points 3.(b) et 4.(d). Faites pour cela des essais avec des histogrammes de tailles croissantes générés aléatoirement.
7. 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

Une petite application implémentant la routine de compression vous est fournie. Une fois compilée, vous pouvez compresser une image au format PGM en un certain nombre de niveaux et la sauvegarder à l'aide de la commande suivante (par exemple pour $k = 4$) :

```
./quantizer lena.pgm 4 compressed.pgm
```

L'application affiche également l'erreur de compression.

Cette application comprend les fichers `main.c`, `ImageQuantizer.h`, `NaiveImageQuantizer.c`, `Reduction.h`, ainsi qu'une bibliothèque de manipulations des images PGM, `PortableGrayMap.c/h` contenant :

- Une fonction `createImageFromFile` qui charge une image au format PGM et la stocke dans une structure `PortableGrayMap`.
- Une fonction `saveImageToFile` qui prend en argument le nom de fichier et une structure `PortableGrayMap` et qui crée un fichier au nom fourni par l'utilisateur contenant l'image au format PGM.
- Une fonction `createEmptyImage` qui permet de créer une image vide aux dimensions données.
- Une fonction `deleteImage` qui permet de libérer la mémoire allouée à une image.

Les images `lena.pgm`, `coins.pgm` et `camera.pgm` vous sont également fournies pour mener vos expériences.

Pour compiler la version naïve, vous pouvez utiliser la commande

```
gcc main.c NaiveImageQuantizer.c PortableGrayMap.c -std=c99 -o quantizer
```

3 Deadline et soumission

Le projet est à réaliser **individuellement** pour le **15 mai 2015** à **23h59** au plus tard. Le projet est à remettre via la plateforme *cicada* : http://cicada.run.montefiore.ulg.ac.be/. Il doit être rendu sous la forme d'une archive `tar.gz` contenant :

(a) Votre rapport (8 pages maximum) au format PDF. Soyez bref mais précis et respectez bien la numération des (sous-)questions.

(b) Le fichier `ImageQuantizer.c`.

(c) Le fichier `GreedyReduction.c`.

(d) Le fichier `DPReduction.c`.

Les implémentations par approche gloutonne et par programmation dynamique seront testées respectivement avec les commandes :

```
gcc main.c GreedyReduction.c ImageQuantizer.c PortableGrayMap.c -std=c99 --pedantic -Wall
-Wextra -Wmissing-prototypes -o quantizer
```

```
gcc main.c DPReduction.c ImageQuantizer.c PortableGrayMap.c -std=c99 --pedantic -Wall -Wextra
-Wmissing-prototypes -o quantizer
```

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 !