Réduction de la palette d'une image

Note préliminaire

Vous rendrez votre travail sur **PROF** sous la forme d'une archive contenant :

- un fichier `README` indiquant les auteurs du travail, une description du travail réalisé (qu'est-ce qui marche, comment vous avez vérifié que cela marche effectivement, etc.) ainsi que vos réponses aux questions théoriques,
- le fichier `LossyGreyQuantizer.java`,
- tous les fichiers de tests que vous aurez écrits.

En particulier, vous ne rendrez pas tous les fichiers qui vous sont fournis !

Le travail à effectuer est dénoté ainsi :

> Écrire une fonction …

Présentation

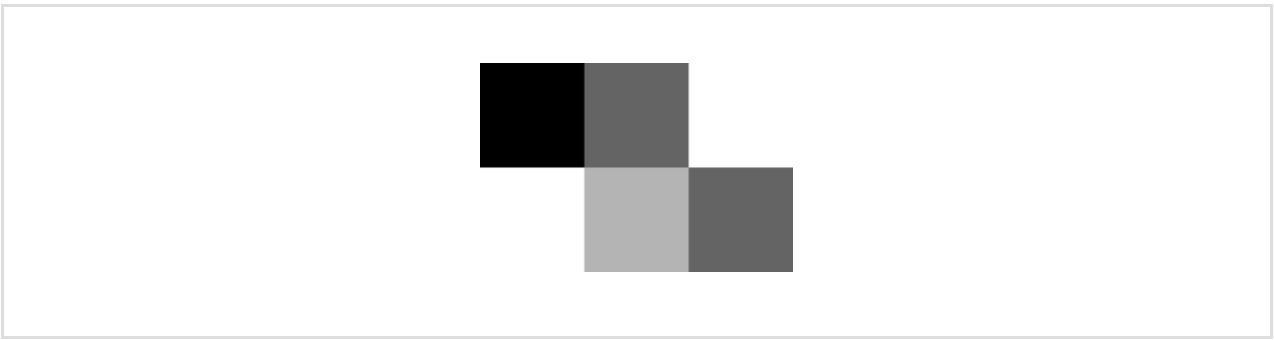
On appelle *réduction de l'espace des couleurs* l'opération qui transforme une image donnée en une image ayant au plus *k* couleurs différentes tout en conservant l'image la plus *proche* possible de l'originale. Idéalement, cette opération supprime essentiellement des détails de l'image, afin de faciliter ou d'améliorer les performances d'opérations ultérieures, qu'il s'agisse de compresser l'image (une grande partie des informations ont disparu, donc il y a moins d'informations à enregistrer) ou d'effectuer des traitements automatiques de l'image (identification des zones de l'image, tri des images, etc.).

N.B. : cette réduction de l'espace est tout aussi valable et intéressante sur les images réellement en couleur (rouge, vert, bleu) qu'en niveaux de gris. Cependant l'algorithme que l'on propose d'utiliser ici fonctionne en niveaux de gris, notamment pour rester simple et conserver un résultat pertinent.

Palette

On parle de *palette* pour l'ensemble des niveaux de gris qui sont utilisés dans une image. Une palette permet de stocker de manière plus efficace une liste de pixels.

Par exemple, l'image



peut être représentée directement à l'aide des niveaux de gris

0	100	255
255	180	100

ou en passant par la palette suivante

code couleur	0	1	2	3
niveau de gris	0	100	180	255

ce qui donne une nouvelle représentation de l'image qui utilise l'indice du niveau gris dans la palette[1]

0	1	3
3	2	1

On vous donne plus loin les outils informatiques pour manipuler des palettes et des niveaux de gris.

Réduction de palette

La réduction de l'espace de niveaux de gris peut se faire en réduisant simplement la palette : à une palette de α niveaux distincts $n_0, n_1, \ldots, n_{\alpha-1}$, tels que $n_i \leq n_{i+1}$, on associe une palette de β niveaux $m_0, m_1, \ldots, m_{\beta-1}$ ($\beta \leq \alpha$) en « fusionnant » plusieurs niveaux de la palette d'origine.

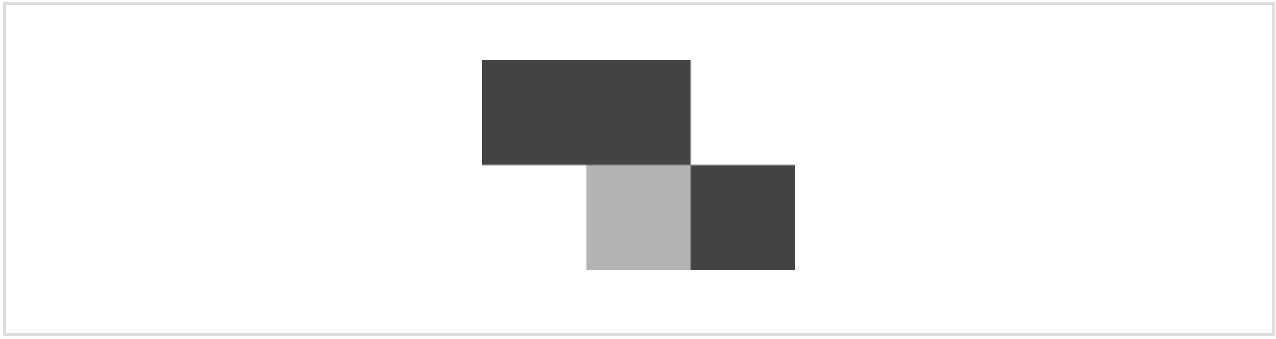
Si on décide de réduire à 3 niveaux de gris la palette à 4 niveaux de l'exemple précédent, on décidera peut-être que la meilleure solution est de fusionner les deux premiers (0 et 100) et de leur associer la valeur 67. On obtient ainsi la palette suivante

code couleur	0	1	2
niveau de gris	67	180	255

La transformation de l'image correspondant à cette réduction de palette est alors simplement l'image dont tous les pixels ont le niveau de gris de la palette réduite associé au niveau de gris de la palette de l'image d'origine. Ainsi, dans l'exemple, tous les pixels de gris 0 ou 1 se retrouvent de gris 0 dans l'image finale ; et tous les pixels de gris 2 ou 3 se retrouvent, respectivement, 1 ou 2. Donc l'image devient

0	0	2
2	1	0

soit, visuellement,



Erreur de la réduction

L'*erreur* associée à cette réduction de palette doit rendre compte de la perte d'information qui a eu lieu. On considèrera ici une erreur assez simple : pour chaque pixel, le carré de la différence de niveaux de gris entre l'image initiale et l'image finale ; pour l'image complète la somme des erreurs pour chaque pixel.

En calculant l'histogramme de l'image qui associe à chaque niveau de gris de la palette le nombre de pixels dudit niveau, proposer le niveau de gris idéal pour réduire la palette à ce seul niveau tout en minimisant l'erreur.
Écrire ainsi la méthode `histogram` de la classe `LossyGreyQuantizer`.

On utilise ici une palette dont les niveaux de gris sont **triés** (en ordre croissant ou décroissant).

Soit un intervalle de $[0\,..\,\alpha - 1]$; on appellera erreur de l'intervalle, l'erreur minimale obtenue en codant les niveaux de l'intervalle de la palette sur une seule couleur.

Justifier que le problème revient à partager la palette complète $[0\,..\,\alpha - 1]$ en β intervalles de façon à minimiser la somme des erreurs des intervalles.

Écrire les méthodes `intervalIdealGrey` et `intervalError` de la classe `LossyGreyQuantizer`. `intervalError` calcule le coût associé à la fusion de toutes les couleurs de l'intervalle $[i\,..\,j]$ de la palette en une seule couleur. `intervalIdealGrey` calcule le niveau de gris à choisir pour minimiser l'erreur lors de la fusion de toutes les couleurs de l'intervalle $[i\,..\,j]$.

Exprimer le problème de réduction de la palette comme une suite des choix des intervalles et en déduire l'équation de récurrence.

Implémenter le calcul de la « meilleure » image en β couleurs en utilisant la programmation dynamique (ce sera la méthode `quantizer`).

Compression

Comme annoncé au début de ce TP, la réduction de la palette permet normalement de réduire la taille du fichier pour peu que l'on enregistre les données dans un format qui prenne ça en compte (ce n'est pas le cas de PGM, ce format n'utilise pas de palette et est conçu pour être le plus simple possible, non pas le plus compact).

> Évaluer le gain de taille d'une image de n pixels en passant d'une palette de α couleurs à une palette de β. On pourra prendre des puissances de 2 pour α et β.

De façon plus pratique, vous pouvez convertir votre image à un format comme PNG avant et après réduction de la palette pour obtenir un gain expérimental. Le format PNG est très souple et offre une large diversité de techniques de compression. Utilisez par exemple :

```
for i in `seq 0 9`; do
    convert -quality 10$i image.pgm image10$i.png;
done
```

pour convertir votre image `image.pgm` au format PNG en utilisant 10 techniques différentes de compression de l'image. Ces résultats donnent une indication du gain d'espace que l'on peut obtenir en pratique en réduisant la palette.

Éléments fournis pour faciliter l'implémentation

On vous propose une **archive** contenant de nombreux éléments pour faciliter votre implémentation.

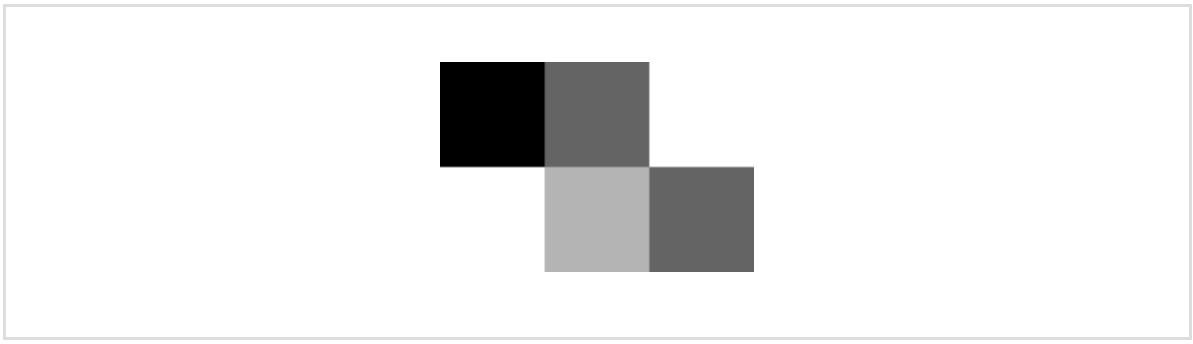
On vous donne notamment `images.jar`, une bibliothèque (rudimentaire afin qu'elle soit le plus simple possible) de manipulation d'images permettant de définir des images et de les charger ou sauver au format PGM (images en niveaux de gris). Les images sont paramétrées par leur largeur (width), hauteur (height) et profondeur (depth), cette dernière correspondant à la valeur maximale de gris disponible pour l'image et la palette (ainsi, si la profondeur est 255, le niveau de gris 0 représentera le noir et 255 le blanc et seules les valeurs comprises entre ces deux bornes seront admissibles).

La bibliothèque propose deux implémentations différentes des images (dans les deux cas, les images sont codées tout simplement comme des tableaux d'entiers, chaque entier correspondant à un pixel, placés dans l'ordre usuel de l'écriture : de gauche à droite et de haut en bas) :

- les images avec palette : chaque cellule du tableau codant l'image indique la couleur de la palette pour le pixel ; ainsi le code

```
int[] palette = new int[] { 0, 100, 180, 255 };
int[] image   = new int[] { 0, 1, 3, 3, 2, 1 };
IndexedGreyImage greyImage = new IndexedGreyImage(3, 2, 255, palette, image);
```

génère l'image



- les images par « composantes » : chaque cellule du tableau codant l'image indique directement le niveau de gris du pixel ; ainsi le code

```
int[] image2  = new int[] { 0, 100, 255, 255, 180, 100 };
ComponentGreyImage greyImage2 = new ComponentGreyImage(3, 2, 255, image2);
```

représente exactement la même image que ci-dessus, en mode par composantes.

Par ailleurs, cette bibliothèque fournit quelques méthodes différentes suivant ces modes, par exemple pour créer de grandes images par collage de petites images les unes à côté des autres (ce qui est pratique pour comparer l'effet d'un paramètre). Explorez la **documentation** pour en savoir plus. Les classes les plus intéressantes sont `PNM` pour lire et écrire des fichiers `.pgm`, `IndexedGreyImage` pour manipuler des images à palette en niveaux de gris et `ComponentGreyImage` pour manipuler des images en niveaux de gris sans palette (en particulier, c'est le résultat obtenu en lisant un fichier `.pgm`). Regardez notamment les constructeurs de ces classes qui offrent notamment des opérations de conversion intéressantes.

On vous fournit également :

i. `LossyGreyQuantizer.java`, un canevas pour votre solution ;

ii. `ImageQuantizationCollage.java` une classe contenant une fonction `main` qui illustre une façon d'utiliser la bibliothèque fournie pour gérer une image, en réduire la palette à différentes tailles et faire un grand collage des résultats ; cette fonction `main` appelle une méthode statique `LossyGreyQuantizer.quantizer` que vous allez écrire pour effectuer le travail proprement dit ; j'ai obtenu le résultat suivant :

iii. de nombreux exemples sur lesquels tester vos algorithmes ; tous les fichiers d'exemple `test-...` ne font que quelques pixels : ouvrez-les dans un éditeur de texte et calculez à la main le résultat que vous devriez obtenir ;

iv. une classe de test et un script permettant d'effectuer les tests sur tous les exemples, ainsi que le résultat que devrait donner ce script ; n'hésitez pas à rajouter (et rendre) d'autres exemples de tests, en particulier pour effectuer des tests des fonctions élémentaires, puisqu'on ne vous fournit des tests que pour la méthode la plus compliquée `quantizer`.

Références

Un algorithme un peu moins naïf de réduction de palette est décrit dans l'article **Grey level reduction for segmentation, thresholding and binarisation of images based on optimal partitioning on an interval**, Quweider, M.K., Scargle, J.D. and Jackson, B.

Par ailleurs, vous pouvez récupérer le code source de la bibliothèque de fonctions (et proposer des patchs, etc.) en utilisant **`git`** :

```
git clone http://www.fil.univ-lille1.fr/~hym/e/java.images/fr.univlille.media.git
```

---

i. On utilise une représentation en tableau 2D de pixels pour coller à la représentation graphique, on codera bien sûr l'image avec un unique vecteur.↩