CY TECH

GRANDE ÉCOLE DES SCIENCES DE L'INGÉNIERIE

Compression d'image : Implémentation d'un Quad-Tree

Rapport de programmation fonctionnelle

Lilian NARETTO

Paul LE DILY

Bilal TAALBI

Corentin BRILLANT

5 avril 2021

Table des matières

Chapitre 1

La méthode Quad-Tree

1.1 La structure d'arbre

Pour utiliser l'aglorithme du Quad Tree qui consiste à transformer la matrice de l'image sous la structure d'arbre, il nous a fallu implémenter une structure d'arbre.
En scala, nous avons donc construit un trait Tree, une case class Node et une case class Leaf qui étendent toutes les deux le trait Tree. La structure récusive suit le procédé suivant :
On regarde la matrice sous la forme de quatre carrés, il est donc préférable que les dimensions soient paires. Chaque noeud de l'arbre est une image et ses quatre fils sont l'image coupée en 4.
Pour chaque fils :
Si tous ses pixels sont de la même couleur, il deviendra une feuille et non un noeud. Sa feuille recevra la couleur des pixels, ainsi que la taille de cette sous-matrice uniforme. La feuille prendra également en attributs les coordonnées de cette sous-matrice associée utiles pour la reconstitution de la matrice totale de l'image.
Sinon le fils est un noeud et on ré-itère le processus.

Dans un noeud, on ordonne les fils dans l'ordre suivant :
- Le premier fils correspond à l'arbre associé à la sous-matrice en haut à gauche dans la matrice associée au père
- Le deuxième fils correspond à l'arbre associé à la sous-matrice en haut à droite dans la matrice associée au père
- Le troisième fils correspond à l'arbre associé à la sous-matrice en bas à droite dans la matrice associée au père
- Le quatrième fils correspond à l'arbre associé à la sous-matrice en bas à gauche dans la matrice associée au père

En notant la matrice M de dimensions m x n, et ses quatre fils M1, M2, M3, M4 , on obtient :
M1 = M[0 :m/2][0 :n/2]
M2 = M[0 :m/2][n/2 :n]
M3 = M[m/2 :m][n/2 :n]
M4 = M[m/2 :m][0 :n/2]

Comment gérer le cas où les dimensions sont impaires ?

Si le nombre de colonnes est impair les deux carrés de gauche auront une colonne de plus que les deux de droite.
Si le nombre de lignes est impair les deux carrés du haut auront une ligne de plus que les deux carrés du bas.

Comment réduire la taille des données ?

Pour compresser l'image davantage qu'elle n'aurait pu l'être avec un quad-tree, nous avons décidé d'hétérogénéiser la couleur des pixels en définissant la différence entre deux. Par exemple on peut considérer que deux pixels avec des composantes (R,G,B) ne sont différents que si le maximum des différences entre chacune de leur composantes ne dépasse pas 50. Prendre l'une ou l'autre de ces deux couleur n'a plus d'importance de ce point de vue. La construction de l'arbre aboutit plus rapidement et surtout la profondeur de l'arbre diminue de manière conséquente.

Nous verrons plus loin comment choisir de manière plus logique les couleurs des pixels lorsqu'on cherche à discrétiser l'espace des pixels possibles pour les composantes RGB.

1.2 La conversion matrice arbre

A l'aide de l'algorithme du Quad Tree, nous pouvons transformer la matrice de l'image en arbre. Il est possible de gagner en compression en suivant la stratégie suivante. L'algorithme du Quad Tree standard nous indique de créer une feuille pour notre arbre lorsque la matrice obtenue sur cette branche ne contient qu'une seule valeur de pixel mais il ne nous dit pas comment considérer que deux pixels représentent la même couleur. Nous avons donc dans notre cas défini la distance entre deux pixels avec la norme 1 (le maximum des différences terme à terme entre chacune de leurs composantes en valeur absolue). A l'aide de cette définition, on peut par exemple décider que deux pixels sont identiques lorsque leur distance est inférieure à une certaine valeur. Pour l'exemple qui suit nous avons établi cette valeur à 50.

FIGURE 1.1 – image de base

FIGURE 1.2 – image après compression Quad Tree

Chapitre 2

Méthodes de modifications d'images

2.1 Inversement des couleurs

Pour inverser les couleurs d'une image et obtenir son négatif nous parcourons la matrice de l'image et à chaque pixel nous y associons la profondeur maximale possible qu'un pixel peut atteindre à laquelle nous soustrayons la valeur originale du pixel en question. Par exemple pour une matrice M de dimensions m x n et dont les pixels sont de profondeur d :
$\forall(i,j) \in [0,m]x[0,n]$,
$a_{i,j} \leftarrow d - a_{i,j}$

2.2 Rotation

Pour tourner une image de 90° vers la droite ou vers la gauche, nous utilisons une fonction qui parcours la matrice en largeur(n) et en longueur(m), et qui renvoie une matrice dans laquelle chaque éléments de la dernière colonne devient la dernière ligne de la matrice, ainsi de suite pour les autres colonnes (processus inverse pour une rotation de droite).
$\forall(i,j) \in [0,m]x[0,n]$,
$droite : a_{j,n-1-i} \leftarrow a_{i,j}$
$gauche : a_{j,i} \leftarrow a_{i,n-1-j}$

2.3 Retournement d'image

Pour retourner une image nous avons utilisé un algorithme qui parcours selon l'axe de symétrie (horizontal ou vertical) la matrice jusqu'à cet axe (c'est-à-dire à moitié). Pour un mirroir vertical nous parcourons la moitié des colonnes de la matrice de l'image et nous l'échangons avec la colonne de l'autre moitié par symétrie par rapport à la ligne située au milieu de la matrice.
Par exemple pour une matrice M de dimensions m x n nous parcourons m/2 lignes et à chaque itération :
$\forall(i,j) \in [0,m]x[0,n]$,
$a_{i,j} \leftarrow a_{m-i,j}$
$a_{m-i,j} \leftarrow a_{i,j}$

De même pour un retournement horizontal sur une matrice de dimensions m x n nous parcourons n/2 colonnes et à chaque itération :
$\forall (i,j) \in [0,m]x[0,n]$,
$a_{i,j} \leftarrow a_{i,n-j}$
$a_{i,n-j} \leftarrow a_{i,j}$

Chapitre 3

Méthodes de classification des couleurs

L'objectif recherché dans le chapitre 3 est de pouvoir classifier les pixels de l'image selon un certain nombre n de couleurs de telle sorte qu'on ait plus que n pixels à renseigner. La matrice de l'image ne contiendra ensuite plus qu'un entier pour chaque pixel c'est-à-dire le numéro de sa couleur, ce qui est plus léger que de renseigner les composantes RGB de chaque pixel. Le problème est donc de trouver les n couleurs optimales parmi celles de l'image de base pour lui être le plus fidèle possible.

3.1 L'algorithme k-means

Ici, nous avons utilisé l'algorithme des k-means. Attention, toutefois, cet algorithme étant relativement coûteux en complexité temporelle, nous nous sommes limités à des images de 256 par 256 pixels. La complexité est fonction du nombre de couleurs souhaité, de la taille de l'image ainsi que du nombre d'itérations que l'on souhaite exécuter. Pour de telles images, nous pouvons aisément réaliser une cinquantaine d'itérations pour cet algorithme (environ 1 seconde pour 5 couleurs et 30 itérations avec nos machines de CYTech). Ci-dessous quelques résultats obtenus :

FIGURE 3.1 – image de base

Image obtenue avec une classification de 2 couleurs :

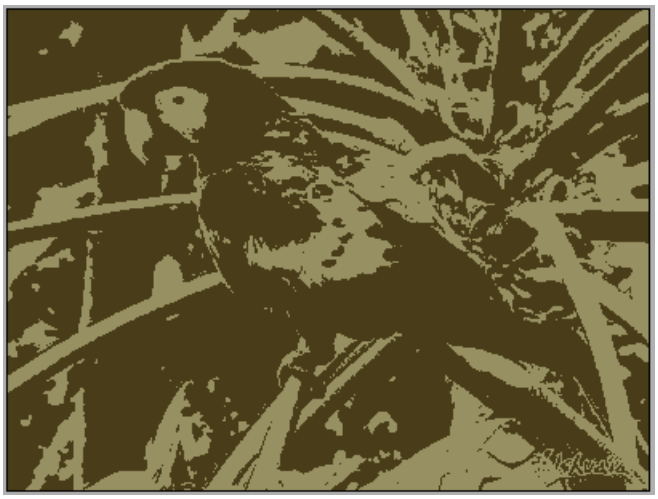

FIGURE 3.2 – image à 2 couleurs

Image obtenue avec une classification de 7 couleurs :

FIGURE 3.3 – image à 7 couleurs

Image obtenue avec une classification de 15 couleurs après une itération du k-means :

FIGURE 3.4 – image à 15 couleurs

Image obtenue avec une classification de 15 couleurs et 10 itérations dans l'algorithme du k-means :

FIGURE 3.5 – image à 15 couleurs pour 10 itérations

Image obtenue avec une classification de 15 couleurs et 30 itérations dans l'algorithme du k-means :

FIGURE 3.6 – image à 15 couleurs pour 30 itérations

3.2 Classification inspirée du codage de Huffman

Nous avons décidé de compresser une image en utilisant une technique s'inspirant de Huffman. Cette technique sort une liste de couleurs qui ont une fréquence d'apparition plus importante. Pour les couleurs qui ont une fréquence d'apparition très faible. la technique était de récupérer les deux couleurs qui ont la fréquence d'apparition la plus faible et d'additioner leur fréquence d'apparition et au lieu de faire une moyenne des deux couleurs nous gardons la couleur avec la fréquence la plus élevée. Le nombre de classes(couleurs retenues) est déterminé arbitrairement.

FIGURE 3.7 – image classifiée selon le codage d'Huffman

Chapitre 4

Compression par codage d'Huffman

Notre implémentation de cet encodage est située dans le package CodageHuffmanPackage. En utilisant le codage d'Huffman sur une profondeur 255 pour chaque composante, on trouve une longueur maximale de 11 bits ce qui est plus long que les 8 bits utilisés classiquement pour trouver un entier entre 0 et 256 exclu. Cependant, en calculant une moyenne de la longueur des codes pondérés par leur fréquence, on obtient une moyenne nettement en dessous du codage classique.

Pour encoder selon le principe de Huffman, on procède de la manière suivante.
On construit un arbre binaire trié selon les fréquences des valeurs de 0 à 255. Chaque feuille correspond à 1 valeur de 0 à 255 et contient la fréquence de cette valeur dans l'image. Pour chaque noeud, il contient la fréquence cumulée obtenue en sommant celles de ses deux fils. La fréquence de son fils gauche est plus élevée que celle de son fils droit. Ensuite, pour obtenir le code d'Huffman de chaque valeur, on redescend depuis la racine vers les feuilles en ajoutant un 0 lorsqu'on part à gauche et un 1 lorsqu'on part à droite. En scala 0 et 1 correspondent à true et false de sorte que chaque code est un Array[Boolean]. Finalement on aplatit la matrice des pixels et on remplace chaque valeur par son code. L'image devient un Array[Boolean].

Chapitre 5

Les packages

La marche à suivre pour compiler l'intégralité des packages est précisée dans le fichier texte README.txt
Le projet contient également une bibliothèque d'image aux formats PBM(P1), PGM(P2) et PPM (P3) pour pouvoir effectuer dfférents tests. Il est possible de générer ses propres images à partir de n'importe quelle image à l'aide du logiciel ImageMagick disponible à https://imagemagick.org/index.php

5.1 Le package ImageReaderWriterPackage

Nous avons décidé d'implémenter notre propre package java pour pouvoir lire et écrire dans des fichiers aux formats P1, P2 et P3 (PBM, PGM, PPM). Nous avons ensuite compilé le package et nous l'avons importé de nos scripts scala. Un exemple d'utilisation du package est donné dans le script principal du projet.

5.2 Le package KmeansPackage

Ce package contient la méthode kmeans

5.3 Le package HuffmanPackage

Ce package contient la méthode huffman

5.4 Le package ImageInterfacePackage

Ce package contient toutes les fonctionnalités de bases du cahier des charges

5.5 Le package HuffmanCodagePackage

Ce package contient toutes les fonctionnalités de bases du cahier des charges