Licence 3 — Nantes Université
Algorithmique & Structures de données 3 — 2025/2026

Projet
Compression d'images bitmap

Consignes importantes :

- Ce projet est obligatoirement réalisé en Java 1.8, obligatoirement par binôme, et obligatoirement écrit en intégralité par le binôme qui le présente. L'utilisation de bibliothèques autres que celles de base (java.lang, java.io, java.util) ou celles utilisées pour les images (voir le fichier ImagePNG.java) est interdite, et vous devez implémenter vous-mêmes toutes les structures arborescentes.
- La complexité de vos algorithmes/programmes est très importante. Seules les solutions assurant une meilleure complexité (calculée comme d'habitude, dans le pire des cas) seront acceptées. Les complexités des opérations requises dans ce sujet, et de toutes les fonctions auxiliaires qu'elles utilisent, seront indiquées en commentaire dans votre code.

1 Description du problème à traiter

Les images bitmap sont composées d'une description de la couleur de chaque pixel. Cette couleur est typiquement codée par ses trois composantes, rouge (R), vert (V) et bleu (B), chacune codée comme un entier dans un octet (8 bits) produisant une palette de $256^3 = 16\,777\,216$ de couleurs. Le principal défaut des images bitmap est leur poids en octets : une image de largeur L et de hauteur H pése a minima [1] $L * H * 3$ octets ; par exemple, le poids d'un fond d'écran 1920×1080 est au minimum de $6\,220\,800\,o \approx 6Mo$. Le format standard PNG [2] utilise un algorithme de compression qui permet de réduire cette taille. Il exploite, entre autres, la présence de zone de couleur identique. Cette compression est sans perte : la couleur de chaque pixel de l'image compressée reste identique à celle de l'image non compressée.

L'**objectif du projet** et de réaliser une compression *avec perte* d'une image bitmap représentée sous forme d'un *R-quadtree*, et d'en observer les effets sur sa *qualité* et son *poids*. La qualité sera mesurée en calculant l'indice EQM (Écart Quadratique Moyen) [3] entre l'image originale et sa version dégradée ; le poids, en *ko*, de l'image PNG dégradée sera comparé à celui du fichier original.

Dans ce projet, chaque couleur sera identifiée par son triplet (R,V,B) (où R, V, B sont des entiers) dans les calculs, et par un code hexadécimal de longueur 6 à l'affichage texte. Ce code hexadécimal correspond aux 24 bits de la couleur. Ainsi, par exemple, la couleur appelée *darkseagreen* [4] a les valeurs suivantes :

— R est égal à 143 en décimal, et à 8f en hexadécimal (et donc à 10001111 en binaire)
— V est égal à 188 en décimal, et à bc en hexadécimal (et donc à 10111100 en binaire)
— B est égal à 143 en décimal, et à 8f en hexadécimal (et donc à 10001111 en binaire)

La couleur darkseagreen est donc représentée par le triplet (143, 188, 143) (correspondant à R, V, B) dans les calculs, et par le code hexadecimal 8fbc8f lorsqu'il faudra l'afficher. Aussi, le blanc est codé (255, 255, 255) ou ffffff, et le noir (0,0,0) ou 000000.

1.1 Représentation d'une image par un *R-quadtree*

Un R-quadtree est un arbre enraciné 4-aire complet (tous les nœuds internes ont degré 4), les fils de chaque nœud étant ordonnés (numérotés de 1 à 4), comme vu en CM et TD : le fils 1 représente le quart Nord-Ouest (NO) de l'image, le 2 le quart NE, le 3 le quart SE et le 4 le quart SO. Un R-quadtree de hauteur maximale n peut être utilisé pour représenter une image de $2^n \times 2^n$ pixels. **Par simplicité, nous**

1. Outre la couleur, d'autres informations (translucidité, ...) sont généralement stockées.
2. https://fr.wikipedia.org/wiki/Portable_Network_Graphics
3. Voir https://fr.wikipedia.org/wiki/Peak_Signal_to_Noise_Ratio
4. http://www.proftnj.com/RGB3.htm

ne traiterons que des images bitmap carrées dont le côté est une puissance de deux. Chaque nœud du R-quadtree à profondeur k (entre 0 et n) représente une région carrée de l'image, de taille $2^{n-k} \times 2^{n-k}$ pixels. Chaque feuille à profondeur n représente ainsi 1 pixel et stocke la couleur exacte de ce pixel.

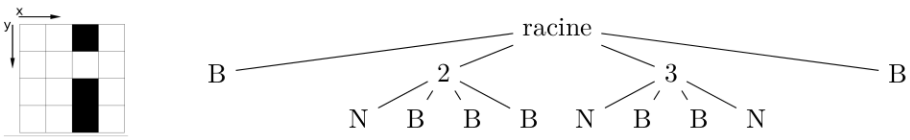

FIGURE 1 – Un R-quadtree représentant la lettre "i" dans une image 4×4 en noir (N) et blanc (B).

Comme vu en CM et TD, un R-quadtree est un arbre de recherche. Appliqué à une image, il permet de mettre en correspondance les coordonnées (x, y) d'un pixel, toujours données à partir de son coin supérieur gauche, et le chemin reliant ce pixel à la racine du R-quadtree. Dans le R-quadtree la Figure 1, la première feuille N à gauche a ainsi pour coordonnées (2, 0) puisque le chemin la reliant à la racine (qui définit l'image entière $[0..3] \times [0..3]$) emprunte d'abord le fils 2 (qui définit la sous-image $[2..3] \times [0..1]$) puis le fils 1 (sous-image $[2] \times [0]$). Cependant, une feuille située à une profondeur inférieure à n représente une zone de plusieurs pixels de couleur homogène. Par exemple, la première feuille B à gauche dans la Figure 1 représente la zone NO entière du R-quadtree (pixels de coordonnées dans $[0..1] \times [0..1]$).

1.2 Compression de *R-quadtree*

Pour compresser avec perte une image PNG, on peut chercher à agrandir au maximum les régions de couleur homogène tout en minimisant la dégradation de l'image. Pour cela, on peut travailler sur le R-quadtree la représentant.

Appelons *sur-feuille* un nœud interne d'un R-quadtree dont les 4 fils sont des feuilles. Par définition d'un R-quadtree, les fils d'une sur-feuille ne sont jamais tous de la même couleur (sinon ce n'est pas un R-quadtree !). Transformer une sur-feuille en feuille en supprimant ses 4 fils (on parle d'*élagage* de l'arbre), et en lui affectant la couleur moyenne de ceux-ci, crée une région 4 fois plus grande de couleur homogène. Afin de minimiser la dégradation de l'image, on peut limiter cet élagage aux sur-feuilles dont les 4 fils ont des couleurs suffisamment proches.

L'œil humain étant particulièrement sensible à la *luminance*[5] L des pixels, la proximité des couleurs peut se calculer à partir de celle-ci : pour une couleur (R, V, B), on définit sa luminance par la formule $L = 0.2126R + 0.7152V + 0.0722B$.

Soit (R_m, V_m, B_m) la couleur moyenne des 4 fils F_1, F_2, F_3, F_4 d'une sur-feuille, définie par la moyenne composante par composante des couleurs (en fait, sa partie entière inférieure), et L_m sa luminance. Plus précisément :

$$X_m = \lfloor \frac{\Sigma_{i \in 1..4} F_i.X}{4} \rfloor, \quad X \in \{R, V, B\} \text{ et } L_m = 0.2126R_m + 0.7152V_m + 0.0722B_m$$

Soient L_1, L_2, L_3 et L_4 les luminances respectives des couleurs des 4 fils de cette sur-feuille. On appelle *dégradation en luminance* λ l'écrat maximum entre L_m et L_i : $\lambda = max_{i \in 1..4} |L_m - L_i|$.

Compression à qualité contrôlée On définit un entier $\Lambda \in 0..255$ qui fixe la dégradation de luminance maximale autorisée. La compression consiste à élaguer *récursivement* toutes les feuilles dont $\lambda \leq \Lambda$. Voir Figure 2 par exemple.

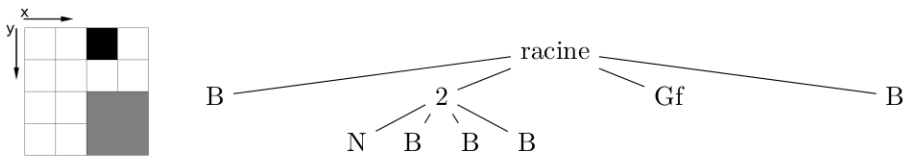

FIGURE 2 – Quadtree de la figure 1 compressé par dégradation en luminance avec $\Lambda = 128$. La couleur $Gf = (127, 127, 127)$ [gris foncé] a remplacé les 4 fils du nœud 3.

5. Voir https://fr.wikipedia.org/wiki/Luminance.

Commpression à poids contrôlé On définit un entier $\Phi > 0$ représentant le nombre maximum de feuilles autorisées dans l'arbre. La compression consiste à élaguer *récursivement* les feuilles par ordre croissant de λ. Voir Figure 3 par exemple.

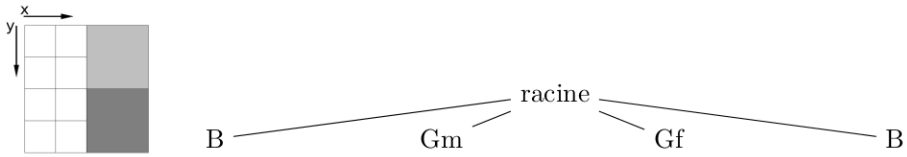

FIGURE 3 – Quadtree de la figure 1 compressé ($\Phi = 4$). Les couleurs $Gm = (191, 191, 191)$ [gris moyen] et Gf ont remplacé les fils des nœuds 2 et 3.

1.3 Représentation de la palette d'une image par un AVL

Les couleurs peuvent être comparées utilisant leurs triplets (R, V, B) et l'ordre lexicographique de ces triplets. Ainsi, $C_1 = (R_1, V_1, B_1)$ et $C_2 = (R_2, V_2, B_2)$ satisfont $C_1 < C_2$ ssi soit $R_1 < R_2$, soit $R_1 = R_2$ et $V_1 < V_2$, soit $R_1 = R_2$ et $V_1 = V_2$ et $B_1 < B_2$.

Par conséquent, on peut stocker toutes les couleurs d'une image PNG dans un AVL. Plus précisément, nous voulons pouvoir :

— parcourir le R-quadtree d'une image et insérer une à une les couleurs de ses feuilles dans un AVL initialement vide (si la couleur n'est pas déjà dans l'arbre).

— pouvoir chercher une couleur donnée par son triplet (R, V, B) et la supprimer à la demande.

2 Rendu de projet

2.1 Dépôt, compilation, exécution

Vous déposerez avant le **samedi 29 novembre à 12h00** sur Madoc une archive (un seul dépôt par binôme) de type .zip, gzip, .gz ou .tgz qui contiendra un seul répertoire Nom1Nom2 (les deux noms de famille des membres du binôme). Ce répertoire doit contenir :, un répertoire appelé `src` contenant les sources de votre programme, un fichier texte README donnant les instructions d'utilisation de votre programme, et des fichiers de tests (images en entrée du programme).

— La compilation doit se faire, une fois l'archive extraite dans le répertoire courant (quel qu'il soit), avec la commande :
`javac -source 1.8 -target 1.8 -d Nom1Nom2/bin Nom1Nom2/src/*.java`

— L'exécution doit se faire avec la commande :
`java -classpath Nom1Nom2/bin Main [args]`
où [args] est une liste d'arguments (le fichier en entrée et d'autres paramètres, voir ci-dessous). Cela signifie que le nom de votre classe contenant le programme principal (`main`) doit être Main.java.

2.2 Implémentation

Les programmes déposés doivent fonctionner parfaitement sur les machines du CIE, sous Linux. Chaque programme doit être intégralement écrit par le binôme qui le dépose[6]. L'appel à des structures de données optimisées existantes (du genre `TreeSet` ou `HashMap`, `HashSet` etc.) est interdit : vous devez implémenter vous-mêmes vos structures de données. Seuls les tableaux (`ArrayList`) et les listes chaînées (`LinkedList`) peuvent être utilisés tels que fournis par Java. Aucun appel à une méthode mise à disposition dans les bibliothèques Java (par exemple un tri d'une liste) n'est autorisé si vous ne connaissez pas le fonctionnement de la méthode et sa complexité (ce que vous montrerez dans les calculs de complexité qui vous sont demandés).

6. Toute ressemblance - même mineure - entre deux programmes, ou entre un programme et du code sur Internet, sera considérée comme non-fortuite et entraînera automatiquement la note de 0 sur la partie concernée ainsi que sur toute autre partie utilisant, même très peu, la partie concernée. Une pénalité supplémentaire de 5 points sera également appliquée. Vous devez prendre des mesures pour vous assurer que votre code ne ressemble à aucun autre et n'est pas partagé, avec ou sans votre accord.

Votre implémentation s'appuiera sur le code source de la classe `ImagePNG` fourni sur Madoc, permettant de charger/sauver/manipuler une image bitmap au format PNG en mémoire, et de calculer l'indice EQM entre deux images. Le programme principal fourni avec cette classe illustre son utilisation. **Vous n'avez pas besoin de comprendre l'implémentation de cette classe, et vous ne devez en aucun cas la modifier**. Outre la classe `ImagePNG`, vous trouverez sur Madoc un échantillon d'images PNG de tailles variables pour tester votre programme (fichier `pngs.zip`).

Vous programmerez une classe `RQuadtree` qui disposera (au moins) des fonctionnalités suivantes :

- un constructeur prenant une `ImagePNG` en paramètre et construisant le R-quadtree correspondant ;
- une méthode `compressLambda` prenant un entier $\Lambda \in 0..255$ en paramètre et appliquant la compression à qualité contrôlée décrite section 1.2 (page 2) ;
- une méthode `compressPhi` prenant un entier $\Phi > 0$ en paramètre et appliquant la compression à poids contrôlé décrite section 1.2 (page 3) ;
- une méthode `toPNG` produisant une `ImagePNG` à partir du R-quadtree ;
- une méthode `toString` produisant la représentation textuelle du R-quadtree sous forme parenthésée, où chaque couleur sera représentée par son code hexadécimal (fourni par la fonction `ImagePNG.colorToHex`). Par exemple, les R-quadtrees des Figures 1, 2 et 3 seront ainsi représentés par les chaînes suivantes :
  1. (fffff (000000 fffff fffff fffff) (000000 fffff fffff 000000) fffff)
  2. (fffff (000000 fffff fffff fffff) 7f7f7f fffff)
  3. (fffff bfbfbf 7f7f7f fffff)

Vous programmerez également une classe `AVL` qui disposera (au moins) des fonctionnalités suivantes :

- un constructeur prenant une `ImagePNG` en paramètre et construisant l'AVL correspondant ;
- un constructeur prenant un `RQuadtree` en paramètre et construisant l'AVL correspondant ;
- trois méthodes `search`, `add` et `remove` prenant en paramètre une couleur (R,V,B) et réalisant les opérations de recherche, ajout et retrait sur l'AVL.
- une méthode `tostring` produisant la représentation textuelle de l'AVL sous forme parenthésée. Chaque couleur sera représentée par son code hexadécimal.

Ces classes pourront utiliser des fonctions et des structures de données auxiliaires, et d'autres classes que vous aurez définies si cela permet un gain de performance (à justifier en commentaires dans votre code).

Vous écrirez aussi un programme principal qui permettra, via un menu textuel, d'accéder à chacune des fonctionnalités suivantes sur un unique `RQuadtree` et un unique `AVL` gérés en mémoire :

1. Construire le R-quadtree correspondant à une image donnée par son nom de fichier PNG
2. Appliquer une compression Λ (pour un Λ donné) au R-quadtree
3. Appliquer une compression Φ (pour un Φ donné) au R-quadtree
4. Sauvegarder l'image représentée par le R-quadtree dans un fichier PNG
5. Sauvegarder la représentation textuelle du R-quadtree dans un fichier TXT
6. Donner les mesures comparatives de deux fichiers images PNG : rapport des poids et indice EQM
7. Construire l'AVL des couleurs correspondant à une image données par son nom de fichier PNG
8. Construire l'AVL des couleurs correspondant au R-quadtree
9. Sauvegarder la représentation textuelle de l'AVL dans un fichier TXT.
10. Effectuer la recherche, l'ajout ou la suppression (3 sous-menus) dans l'AVL d'une couleur donnée en hexadécimal.

Le programme pourra aussi être exécuté en mode non-interactif en l'invoquant avec en paramètre de ligne de commande le nom d'un fichier PNG, une chaîne indiquant la méthode de compression à utiliser ("Lambda" ou "Phi") et le paramètre pour cette méthode (un entier). Il appliquera alors automatiquement la compression indiquée à l'image donnée en produisant tous les fichiers de sortie associés et en affichant les mesures de qualité. Par exemple, appliqué à un fichier nommé `i.png` avec la méthode "Lambda" et l'entier 20, il produira les fichiers `i-lambda20.png`, `i-lambda20R.txt`, `i-lambda20AVL.txt` correspondant à la compression Lambda sous forme d'image et sous forme textuelle, ainsi qu'à l'AVL des couleurs ; il affichera aussi le rapport de poids et l'indice EQM entre `i.png` et `i-lambda20.png`.