Projet
Compression d'images bitmap

Consignes importantes : ce projet est obligatoirement réalisé en Java, obligatoirement par binôme, et obligatoirement écrit en intégralité [1] par le binôme qui le présente.

1 Description du problème à traiter

Les images bitmap sont composées d'une description de la couleur de chaque pixel de l'image, typiquement codée par ses trois composantes, rouge (R), vert (V) et bleu (B), chacune sur un octet, produisant une palette de $256^3 = 16\,777\,216$ de couleurs. Le principal souci est leur poids : une image de largeur L et de hauteur H pèse a minima [2] $L*H*3$ octets ; par exemple, le poids d'un fond d'écran 1920×1080 est au minimum de $6Mo$. Le format standard PNG (https://fr.wikipedia.org/wiki/Portable_Network_Graphics) utilise un algorithme de compression qui permet de réduire cette taille. Il exploite, entre autres, les zones de couleur identique. Cette compression est toutefois 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 enregistrée au format PNG et d'en observer les effets sur sa *qualité* et son *poids*. Pour ce faire on utilisera une représentation de l'image sous forme d'un *quadtree*.

1.1 Représentation d'une image par un *quadtree*

Un R-quadtree (par abus de langage, on parlera simplement de *quadtree* dans la suite du sujet) est un arbre enraciné 4-aire complet (tous les nœuds internes ont degré 4), les fils de chaque nœud étant ordonnées (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 quadtree de hauteur n peut être utilisé pour représenter une image de $2^n \times 2^n$ pixels. **Par simplicité, nous ne traiterons que des images bitmap carrées dont le côté est une puissance de deux.** Chaque nœud à profondeur k (entre 0 et n) représente une région carrée de l'image, de taille $2^{n-k} \times 2^{n-k}$ pixels. Un nœud à profondeur n représente ainsi 1 pixel et stocke la couleur exacte de ce pixel.

On peut tout d'abord se représenter une image comme un *quadtree plein* : les feuilles d'un tel quadtree sont toutes à profondeur n et chacune représente donc un unique pixel. La Figure 1 illustre un quadtree plein représentant une image 4×4. Ce quadtree plein a donc toutes ses feuilles à profondeur 2 ($= log_2 4$), chacune représentant un pixel noir (N=$(0,0,0)$) ou blanc (B=$(255,255,255)$).

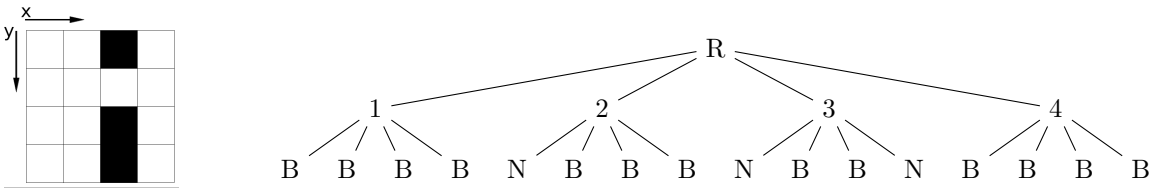

FIGURE 1 – Quadtree représentant la lettre "i" dans une image 4×4.

Comme vu en CM et TD, un quadtree (plein ou non) 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 quadtree. Dans le quadtree plein de 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 $[2] \times [0]$).

1. Cf. consignes détaillées sur Madoc.
2. Outre la couleur, d'autres informations (translucidité, ...) sont généralement stockées.

1.2 Compression de *quadtree*

Pour gagner de la place, il est possible de réduire l'arbre par élagage (élimination de feuilles) : si toutes les feuilles d'un même nœud ont la même couleur, on peut alors supprimer ces quatre feuilles ; on fait ainsi diminuer l'arbre (gain d'espace mémoire) et on rend l'accès à la couleur d'un pixel plus rapide (gain de temps). Cette compression se fait sans perte, i.e., l'image n'est pas dégradée.

Par exemple, la Figure 2 illustre la compression sans perte du quadtree plein de la Figure 1. Noter que les nœuds compressés ont pris la couleur de leurs fils supprimés. Noter enfin qu'il s'agit alors d'un quadtree (plus exactement R-quadtree) tel que vu en CM et TD.

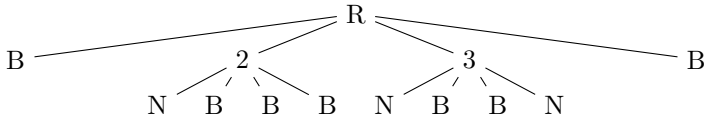

FIGURE 2 – Quadtree de la figure 1 compressé (sans dégradation).

En autorisant un certain niveau de perte, on peut réduire l'arbre plus fortement. Nous proposons deux méthodes s'appuyant toutes deux sur *l'écart colorimétrique* : soient $F_1, \ldots, F_4$ les quatre feuilles d'un même nœud N. On note (R_i, V_i, B_i) la couleur de chaque feuille F_i. La couleur moyenne (R_m, V_m, B_m) de ces feuilles est alors définie comme suit :

$$R_m = \frac{\sum_{i\in 1..4} R_i}{4}\ , \quad V_m = \frac{\sum_{i\in 1..4} V_i}{4}\ , \quad B_m = \frac{\sum_{i\in 1..4} B_i}{4}\ .$$

L'écart colorimétrique λ_i de la feuille F_i est sa distance normalisée à la couleur moyenne :

$$\lambda_i := \sqrt{\frac{(R_i - R_m)^2 + (V_i - V_m)^2 + (B_i - B_m)^2}{3}},$$

et l'écart colorimétrique Λ au niveau du nœud N est défini par :

$$\Lambda = max_{i\in 1..4}\ \lambda_i.$$

On considère deux méthodes de compression, l'une basée sur une qualité minimale, l'autre sur un poids maximal :

1. On peut définir une dégradation maximale autorisée sous la forme d'un entier $\Delta \in 0..192$: les feuilles $F_1, \ldots, F_4$ d'un même nœud N sont supprimées ssi $\Lambda \leq \Delta$. Voir Figure 3 par exemple.

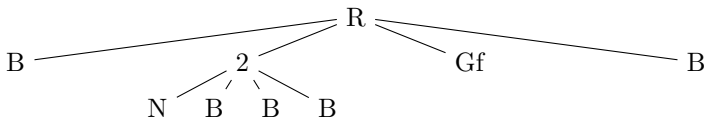

FIGURE 3 – Quadtree de la Figure 2 compressé ($\Delta = 128$). La couleur $Gf = (127, 127, 127)$ [gris foncé] a remplacé les 4 fils du nœud 3.

2. On peut également définir la dégradation en fonction d'un poids maximum souhaité, ce qui peut se traduire par un entier $\Phi > 0$ représentant le nombre maximum de feuilles autorisées dans l'arbre. Les feuilles sont alors élaguées par ordre croissant de Λ. Voir Figure 4 par exemple.

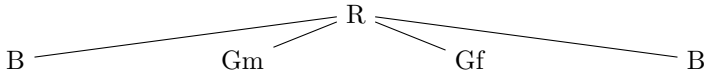

FIGURE 4 – Quadtree de la Figure 2 compressé ($\Phi = 4$). Les couleurs $Gm = (191, 191, 191)$ [gris moyen] et Gf ont remplacé les fils des nœuds 2 et 3.

2 Travail demandé

Le projet devra être rendu avant le **samedi 5 décembre à 23h59** [3] par dépôt d'une archive ZIP sur Madoc. L'archive sera intitulée aux noms des membres du binôme. Elle comprendra :

— le code source documenté de votre projet ;
— un fichier texte donnant les instructions de compilation et d'utilisation de votre programme (qui doit compiler et être utilisable en dehors d'Eclipse ou tout autre IDE) ;
— votre rapport (environ 10 pages) au format PDF.

2.1 Implémentation

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. Le programme principal fourni avec cette classe illustre son utilisation. **Vous n'avez pas besoin de modifier cette classe, ni de comprendre l'implémentation de ses méthodes.** 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 `Quadtree` qui disposera (au moins) des fonctionnalités suivantes :

— un constructeur prenant une `ImagePNG` en paramètre et construisant le quadtree correspondant ; le quadtree obtenu devra être compressé sans dégradation (cf. Figure 2) ;
— une méthode `compressDelta` prenant un entier $\Delta \in 0..255$ en paramètre et appliquant la première technique de compression décrite ci-avant (cf. Figure 3) ;
— une méthode `compressPhi` prenant un entier $\Phi > 0$ en paramètre et appliquant la seconde technique de compression décrite ci-avant (cf. Figure 4) ;
— une méthode `toPNG` produisant une `ImagePNG` à partir du quadtree.
— une méthode `toString` produisant la représentation textuelle du quadtree sous forme parenthésée, où chaque couleur sera représentée par son code hexadécimal (fourni par la fonction `ImagePNG.colorToHex`).
Par exemple, les quadtrees des Figures 1 à 4 seront ainsi représentés par les chaînes suivantes :

```
1. ((ffffff ffffff ffffff ffffff) (000000 ffffff ffffff ffffff) (000000 ffffff ffffff 000000)
   (ffffff ffffff ffffff ffffff))
2. (ffffff (000000 ffffff ffffff ffffff) (000000 ffffff ffffff 000000) ffffff)
3. (ffffff (000000 ffffff ffffff ffffff) 7f7f7f ffffff)
4. (ffffff bfbfbf 7f7f7f ffffff)
```

Ces méthodes 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. Cela sera justifié dans le rapport.

Vous écrirez aussi un programme principal qui permettra, via un menu textuel, d'accéder à chacune des fonctionnalités suivantes :

1. Charger une image PNG en mémoire dans un quadtree
2. Appliquer une compression Delta pour un Δ donné
3. Appliquer une compression Phi pour un Φ donné
4. Sauvegarder le quadtree dans un fichier PNG
5. Sauvegarder la représentation textuelle du quadtree dans un fichier TXT
6. Donner les mesures comparative de deux fichiers images PNG

Le programme pourra aussi être exécuté en mode non-interactif en l'invoquant avec en paramètres le nom d'un fichier PNG, un entier Δ et un entier Φ. Dans ce cas, il appliquera automatiquement tous les traitements à l'image donnée en produisant tous les fichiers de sortie associés et en affichant les mesures pour chacune des compressions appliquées. Par exemple, appliqué à un fichier nommé `i.png`, et pour des valeurs $\Delta = 100$ et $\Phi = 4$, il produira les fichiers `i-delta20.png`, `i-delta20.txt`, `i-phi4.png`, `i-phi4.txt` correspondant respectivement aux compressions Delta et Phi, sous forme d'image et sous forme textuelle ; il affichera aussi le rapport de poids et l'indice EQM [4] entre `i.png` et `i-delta20.png` et entre `i.png` et `i-phi4.png`.

3. Le lien de dépôt sera désactivé passé cette heure limite.
4. l'Écart Quadratique Moyen (cf. https://fr.wikipedia.org/wiki/Peak_Signal_to_Noise_Ratio) permet de mesurer la qualité relative d'une image compressée vis à vis de l'originale. Son calcul est fourni dans la classe `ImagePNG` (voir programme principal dans cette classe).

2.2 Rapport

Votre rapport comportera les parties suivantes :

1. Une introduction présentant les objectifs du projet dans le cadre du module (selon vous).
2. Une présentation des algorithmes que vous avez mis en œuvre pour chaque méthode de la classe `QuadTree` que vous avez implémentée. Vous discuterez en particulier les choix importants que vous avez faits pour leur implémentation et les structures de données auxiliaires utilisées (s'il y en a).
3. Une analyse de la complexité temporelle de ces algorithmes (en ordre de grandeur).
4. Une conclusion en forme de discussion critique sur votre travail et les extensions ou variations envisageables, avec une estimation de leur impact en terme de complexités spatiale et temporelles (par exemple, d'autres structures d'arbres, ...).

La présentation et la rédaction du rapport seront prises en compte dans son évaluation.

2.3 Démonstration

Votre programme devra être opérationnel pour votre dernière séance de TP encadré sur le projet, semaine 49 (30/11-4/12) ; vos chargés de TP passeront vérifier avec vous son fonctionnement au cours d'une courte démonstration ($\sim$ 5 minutes) durant cette séance.