ASD3 - Projet
Arbres et compression d'image

David JULIEN Vincent KOWALSKI

31 octobre 2023

1 Introduction

Le dernier rapport de Sandvine [1] indique qu'en 2022, la majeure partie (jusqu'à 76% du débit descendant et 40% du débit montant selon les régions) du trafic internet mondial était un flux vidéo, comme illustré dans la Figure 1 :

Regional Category Traffic Share Upstream

AMERICAS CATEGORY SHARE — UPSTREAM TRAFFIC

	Category	Total Volume
1	Video	39.96%
2	Cloud	20.39%
3	Web Browsing	11.76%
4	Messaging	9.39%
5	File Sharing	4.45%
6	VPN	4.39%
7	Marketplace	3.19%
8	Gaming	3.03%
9	Social Networking	2.91%
10	Audio	0.53%

APAC CATEGORY SHARE — UPSTREAM TRAFFIC

	Category	Total Volume
1	Video	31.91%
2	Cloud	22.97%
3	File Sharing	13.90%
4	Messaging	9.09%
5	Web Browsing	7.24%
6	VPN	4.73%
7	Gaming	3.35%
8	Marketplace	3.31%
9	Social Networking	3.16%
10	Audio	0.35%

EMEA CATEGORY SHARE — UPSTREAM TRAFFIC

	Category	Total Volume
1	Video	39.65%
2	Messaging	12.40%
3	Cloud	10.64%
4	Social Networking	9.93%
5	Web Browsing	9.30%
6	Marketplace	5.47%
7	File Sharing	5.45%
8	VPN	4.95%
9	Gaming	1.96%
10	Audio	0.26%

Regional Category Traffic Share Downstream

AMERICAS CATEGORY SHARE — DOWNSTREAM TRAFFIC

	Category	Total Volume
1	Video	76.43%
2	Gaming	5.99%
3	Marketplace	4.27%
4	Social Networking	3.83%
5	Web Browsing	2.67%
6	Cloud	2.18%
7	Messaging	1.93%
8	File Sharing	1.19%
9	Audio	0.95%
10	VPN	0.55%

APAC CATEGORY SHARE — DOWNSTREAM TRAFFIC

	Category	Total Volume
1	Video	69.45%
2	Gaming	7.72%
3	Marketplace	7.57%
4	Cloud	3.37%
5	Social Networking	3.31%
6	Web Browsing	2.48%
7	File Sharing	2.35%
8	Messaging	1.59%
9	VPN	1.32%
10	Audio	0.85%

EMEA CATEGORY SHARE — DOWNSTREAM TRAFFIC

	Category	Total Volume
1	Video	65.29%
2	Social Networking	14.76%
3	Marketplace	4.04%
4	Messaging	4.01%
5	Web Browsing	3.77%
6	Gaming	3.14%
7	File Sharing	2.23%
8	Cloud	1.57%
9	VPN	0.89%
10	Audio	0.31%

FIGURE 1 – Répartition du trafic internet montant et descendant, selon les régions du monde. "APAC" signifie "Australie, Pacifique" et "EMEA" signifie "Europe, Moyen-Orient, Asie".

Cette demande importante de flux vidéo invite les différents acteurs à optimiser la qualité de leurs algorithmes de compression, c'est-à-dire à minimiser la quantité de données à envoyer (leur objectif) tout en maintenant une qualité minimum (leur contrainte) pour satisfaire leurs client·e·s.

Les questions de compression ne sont toutefois pas réservées aux flux vidéo. En effet, c'est un problème plus général de flux d'images (statiques ou non) : les fichiers d'image transportent beaucoup d'information (par exemple, 4 octets par pixel dans le cadre de la norme RGBA8888 utilisée par le format PNG) dont il est difficile de se passer, la couleur d'un pixel étant a priori décorrellée de celle de ses voisins. Dans ce projet, on se propose de mettre en place des algorithmes de compression d'image au format *Portable Graymap Format*[1] (**PGM** dans la suite), pour réduire la taille d'un fichier représentant une image, au prix d'une baisse de qualité.

2 Projet

Note : Le projet devant être exécutable sur une machine du CIE, toutes les considérations seront faites par rapport à une machine tournant sous Linux.

2.1 Spécification du PGM

Le PGM est un format d'image développé pour faciliter les échanges entre différentes plateformes : il a été inventé pour envoyer des images monochromes par email en ASCII brut, permettant la restitution de l'image quel que soit le format de texte utilisé sur la plateforme. Toutes les informations nécessaires à l'affichage de l'image sont contenues dans le corps d'un fichier texte, c'est-à-dire au début de celui-ci (le cas échéant, après les métadonnées). Pour lire un fichier au format PGM, vous pouvez utiliser n'importe quelle visionneuse d'image ; il vous suffit d'ouvrir le fichier avec votre outil en double-cliquant dessus.

Le *header* est composé de trois informations, comme dans l'Exemple 1 suivant.

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
```

Exemple 1 – Header d'un fichier PGM.

Dans l'Exemple 1, la première ligne correspond au *"Magic number"*, qui détermine le format utilisé (`P1` pour une *bitmap*, `P2` pour une image en niveaux de gris, `P3` pour une image couleur). Dans ce projet, nous travaillerons avec des images en niveaux de gris : les fichiers que vous manipulerez commenceront donc toujours par `P2`. La deuxième ligne est un commentaire. La troisième ligne donne la largeur et la hauteur de l'image, en pixels ; les deux valeurs sont séparées par un espace. Ici, l'image mesure donc 24 pixels de large, et 7 de haut. La quatrième ligne donne la valeur maximale `max` de luminosité d'un pixel (elle doit être inférieure ou égale à 255). Ici, la luminosité maximale est 15, il y a donc 16 niveaux de gris (de 0 à 15).

Les données de l'image (ici, le niveau de gris de chaque pixel) sont ensuite présentées comme une série de nombres, obligatoirement compris entre 0 et `max` et séparés par des espaces et/ou des `\n`. L'Exemple 2 donne le contenu d'une image sous format PGM.

```
P2
# Shows the word "FEEP" (example from Netpbm man page on PGM)
24 7
15
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 14 14 14 14  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 14  0  0 14  0
0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0  0 14 14 14 14  0
0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0 14  0  0  0  0
0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11  0  0 14  0  0  0  0
0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
```

Exemple 2 – Contenu d'un fichier PGM.

1. voir `https://en.wikipedia.org/wiki/Netpbm`

Ici, il est important de noter qu'en dehors des commentaires, le format ne fait aucune distinction entre les espaces et les `\n`, et est insensible à la répétition de ceux-ci. Ainsi, bien que ne comportant qu'une ligne de valeurs séparées par un ou plusieurs espaces, l'Exemple 3 suivant produira la même image.

```
P2 24 7 15 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
     0  0  0 0  3  3  3  3  0  0  7  7  7  7  0  0 11 11 11 11  0  0 14
    14 14 14  0 0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0  0
    14  0  0 14  0 0  3  3  3  0  0  0  7  7  7  0  0  0 11 11 11  0  0
    0 14 14 14 14  0 0  3  0  0  0  0  0  7  0  0  0  0  0 11  0  0  0  0
      0 14  0  0  0  0 0  3  0  0  0  0  0  7  7  7  7  0  0 11 11 11 11
     0  0 14  0  0  0  0 0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0  0
    0  0  0  0  0  0  0  0
```

Exemple 3 – Contenu d'un fichier PGM sur une seul ligne.

2.2 Conversion image → arbre

On souhaite représenter les images PGM à l'aide de *quadtrees*, et les utiliser pour compresser les images de sorte à réduire au maximum l'espace de stockage. Par la suite, il s'agit donc de bien faire la différence entre *le fichier PGM* (qui contient l'image brute) et *la représentation de l'image* (c'est-à-dire le *quadtree* associé).

Soit l'exemple suivant (volontairement simple) de fichier PGM (à gauche) ainsi que sa visualisation (agrandie 20 fois, à droite) :

```
P2
4 4
15
0   1   4   5
2   3   6   7
8   9  12  13
10 11  14  15
```

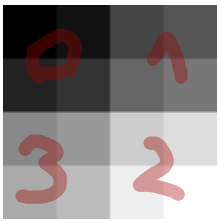

FIGURE 2 – Contenu et visualisation d'un fichier PGM d'exemple.

Dans ce projet, nous utiliserons les *quadtrees* vus en cours. Tout d'abord, nous allons représenter l'image brute dans un *quadtree*, comme dans la Figure 3.

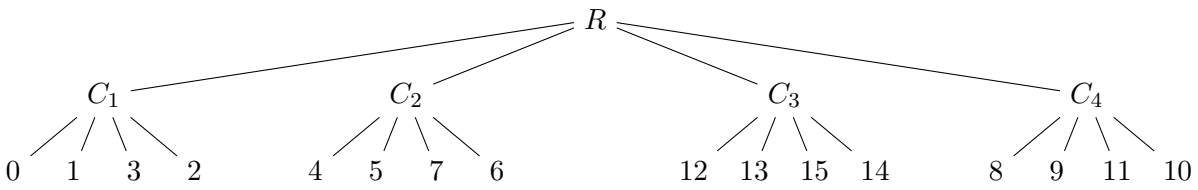

FIGURE 3 – *Quadtree* représentant le fichier PGM précédent.

L'arbre doit donc contenir toutes les informations nécessaires à la reconstruction de l'image, c'est-à-dire la largeur et la hauteur, la luminosité maximale possible ainsi que la luminosité de chaque nœud. Toutefois, on rappelle que le but est de minimiser l'espace mémoire utilisé par les fichiers d'image : il est donc inutile de stocker des informations que l'on peut retrouver dans la structure que l'on utilise, et vous devrez penser à maintenir une structure de *quadtree* conforme aux définitions précises dans le cours. Notamment, il est précisé dans le cours que dans les nœuds comportant 4 enfants feuilles (comme chacun des nœuds C_1, C_2, C_3, C_4 dans la Figure 3) ces 4 feuilles ne peuvent être identiques. Ainsi, l'arbre de la Figure 4 n'est pas un *quadtree* tel que décrit dans le cours.

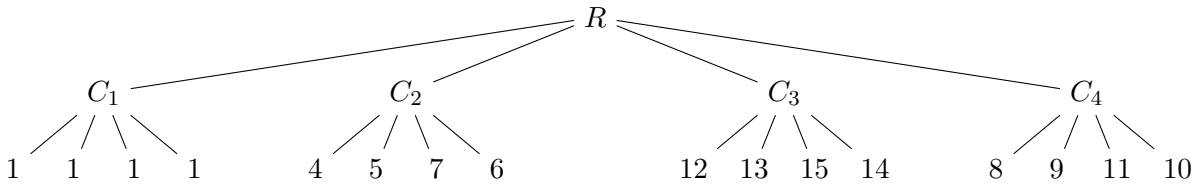

FIGURE 4 – Contre-exemple de *quadtree*.

Note : Les compressions d'images proposées ci-dessous sont *avec perte* : une fois le *quadtree* modifié, l'image est modifiée, et l'image initiale ne peut plus être retrouvée à l'identique.

2.3 Diminution de la représentation en mémoire

2.3.1 Effeuillage (ou compression Lambda)

Une première manière de diminuer la taille du *quadtree* représentant l'image est de regrouper quatre feuilles qui ont le même parent P (on appelera P une *brindille*) sous la forme d'une seule feuille au niveau supérieur (qui prend donc la place de P). Lorsque cette opération est effectuée pour toutes les brindilles de l'*arbre initial*, on peut dans le meilleur des cas (arbre complet comme dans la Figure 3), diviser par 4 le nombre d'éléments dans l'arbre.

Pour mettre en place cette compression, on s'intéresse à la luminosité, notée λ, des nœuds.

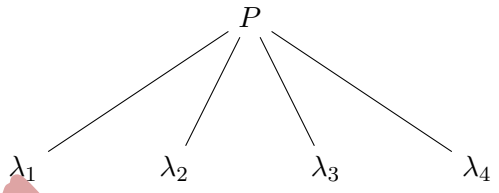

FIGURE 5 – *Brindille* P, à compresser.

Étant donnée une *brindille* (c'est-à-dire quatre feuilles qui ont le même parent P) comme dans la Figure 5, on souhaite calculer la moyenne logarithmique de luminosité Λ pour ces 4 feuilles (dont les luminosités respectives sont notées $\lambda_i, 1 \leq i \leq 4$) donnée par la formule :

$$\Lambda = \exp\left(\frac{1}{4}\sum_{i=1}^{4} \ln(0.1 + \lambda_i)\right). \tag{1}$$

Par définition, la luminosité de la feuille résultante sera la valeur de Λ arrondie à l'entier le plus proche (dénoté par l'appel à la fonction round () dans la suite).

Par exemple, la brindille C_1 dans la Figure 3 est compressée comme dans la Figure 6 car

$$\text{round}\left(\exp\left(\frac{1}{4}(\ln(0.1) + \ln(1.1) + \ln(3.1) + \ln(2.1))\right)\right) = \text{round}\,(0.91990...) = 1.$$

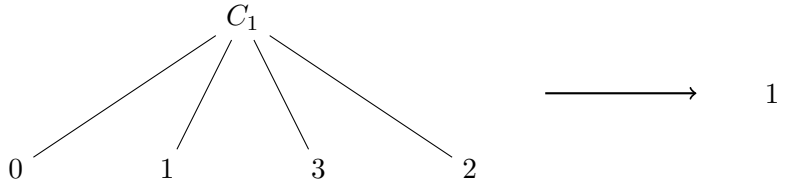

FIGURE 6 – Compression du sous-arbre C_1.

Alors, on peut supprimer cette brindille, et la remplacer par une feuille de luminosité round (Λ) dans le *quadtree* global, comme dans la Figure 7.

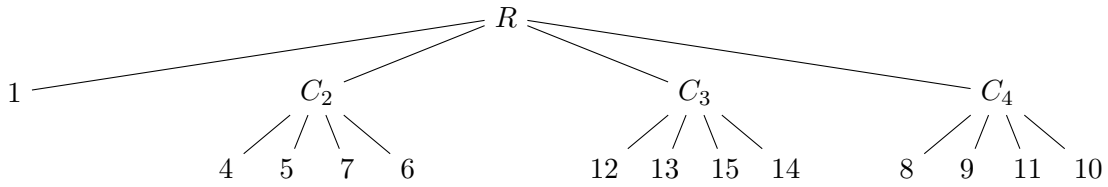

FIGURE 7 – *Quadtree* où la brindille représentée par le nœud C_1 a été compressée.

On réitère le processus pour chacun des nœuds C_2, C_3, C_4 et on obtient la Figure 8.

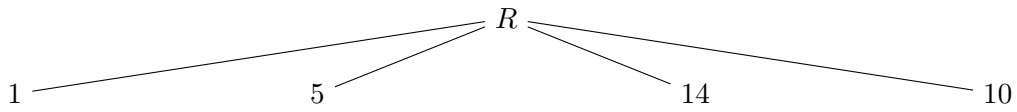

FIGURE 8 – *Quadtree* où toutes les brindilles de l'arbre initial ont été compressées.

Dans cette partie, il vous est demandé de :
- générer le *quadtree* correspondant à une image brute `image.pgm` fournie en entrée (rappel : il n'y a pas d'information a priori sur le nombre de lignes et de valeurs par ligne dans le fichier `pgm` ; vous considérerez tous les cas possibles) ;
- implémenter l'algorithme de compression décrit précédemment pour une brindille ;
- effectuer un effeuillage complet : à partir de l'arbre de départ, générer un nouvel arbre dans lequel on aura compressé toutes les brindilles existantes dans l'arbre de départ. Vous veillerez à maintenir la structure de votre arbre (en évitant les cas similaires à la Figure 4).
- générer l'image correspondante au format PGM.

2.3.2 Compression dynamique (ou compression Rho)

L'algorithme précédent a le désavantage de ne pas évaluer les destructions apportées à une image : les feuilles sont fusionnées, peu importe la quantité d'informationperdue. Dans cette partie, nous allons effectuer une compression similaire, mais en interdisant de perdre trop d'information : si une brindille a une valeur Λ trop éloignée de celle de l'une des feuilles, alors on préférera - si possible - compresser une autre brindille, avec une valeur moins éloignée. On pourra alors se permettre une compression dynamique, c'est-à-dire compresser les zones peu contrastées (avec peu de variations de luminosité) et laisser les zones très contrastées (avec beaucoup de variations) telles quelles. On pourra aussi compresser des feuilles résultant d'une compression précédente.

Pour chaque brindille P existant *à un moment donné* dans l'arbre (donc pas forcément dans l'arbre initial), on calcule l'écart maximum ε entre la moyenne logarithmique de luminosité Λ du nœud P et la luminosité λ_i de l'un de ses enfants :

$$\varepsilon = \max_{1 \leq i \leq 4} |\Lambda - \lambda_i|. \tag{2}$$

La compression globale consiste maintenant à compresser la brindille d'écart ε minimum existant dans le *quadtree* courant (encore une fois, ce doit être un *quadtree* correct), jusqu'à ce que le *quadtree* résultant soit assez petit. Pour cela, une valeur entière ρ entre 0 et 100 est fournie en entrée, et la compression s'arrête dès que le *taux de compression* devient inférieur ou égal à ρ%. Le *taux de compression* se définit comme le ratio entre le nombre de noeuds du *quadtree* courant et le nombre de noeuds du *quadtree* initial. Couleurs différentes

Par exemple, soit $\rho = 82$. Dans l'arbre de la Figure 3, il y a 21 nœuds. On cherche donc à compresser l'arbre jusqu'à ce qu'il ait au plus 17 noeuds. Les écarts maximum de luminosité pour les quatre

brindilles sont (de gauche à droite) :

$$\varepsilon_1 = 2.081\ldots, \quad \varepsilon_2 = 1.514\ldots, \quad \varepsilon_3 = 1.534\ldots, \quad \varepsilon_4 = 1.553\ldots. \tag{3}$$

En sélectionant ε_2, on compressera la brindille C_2, et on obtient le *quadtree* donné dans la Figure 9 qui a 17 noeuds :

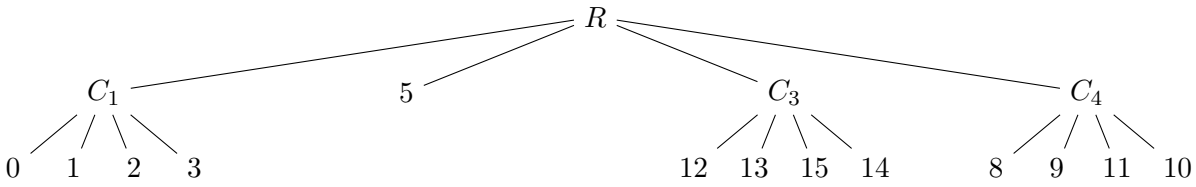

FIGURE 9 – *Quadtree* résultant de la compression dynamique du *quadtree* de la Figure 3, après une compression dynamique avec $\rho = 82$.

Dans cette partie, il vous est demandé de :
— implémenter l'algorithme de compression décrit précédemment pour une brindille ;
— effectuer une passe de compression avec une valeur ρ donnée ;
— générer l'image correspondante au format PGM.

3 Travail à rendre

3.1 Modalités de rendu

— Ce projet est à réaliser en binôme, doit être compilable par `Java 1.7` et parfaitement fonctionnel sur les machines du CIE ;
— Il devra également être indépendant d'un *IDE* comme Eclipse ou VSCode, c'est-à-dire qu'il sera compilable et exécutable en ligne de commande (dans un terminal) ;
— Le binôme doit avoir écrit la totalité du code proposé ; toute ressemblance avec du code extérieur (autre binôme, internet...) entraînera l'annulation des points sur la partie concernée et 5 points supplémentaires de pénalité (et cela pour chaque binôme concerné) ;
— Chacun des algorithmes mis en place doit être *aussi efficace que possible* en termes de complexité au pire, et doit pour cela mettre en oeuvre les structures de données les plus adaptées.
— Vous devez implémenter vous-même les structures de données que vous utilisez : l'utilisation de structures complexes autres que les listes et tableaux (par exemple les `TreeSet, HashMap`, etc...) est prohibée, tout comme celle de librairies externes ; ArrayList<T>; List<T>; Array<T>
— Vous pouvez cependant utiliser les librairies de gestion de fichiers pour lire et écrire dans un système de fichiers ;
— Vous devez joindre un rapport d'un maximum de 12 pages incluant
  — les commandes de compilation et d'exécution en mode console ;
  — une vue globale de votre programme, sous la forme d'un algorithme en pseudo-code, dont les instructions sont des appels à des procédures ; la spécification et les paramètres de chaque procédure sera précisément décrit ;
  — pour chaque méthode décrite en Section 2.3.1 et Section 2.3.2, un détail sous forme algorithmique (en pseudo-code) de son fonctionnement, une explication de celui-ci et un calcul de complexité ;
  — des exemples de résultats obtenus avec votre programme ;
— L'ensemble des fichiers nécessaires à l'exécution et l'évaluation de votre projet (fichiers source, images, rapport...) sera placé dans un répertoire nommé `Nom1Nom2`, qui sera ensuite archivé au format `tar.gz` sous le nom `Nom1Nom2.tar.gz` ou au format `zip` sous le nom `Nom1Nom2.zip`[2] ;

2. Suivez bien les consignes et ne fournissez pas d'archive d'un autre type, renommée (ou non) manuellement. Elle n'ouvrira pas et ne sera pas évaluée.

— L'archive de rendu sera déposée sur Madoc, au plus tard le **dimanche 10 décembre à 23h59 heure de Paris (UTC+1)**. **Attention** : Madoc n'acceptera aucun retard !

3.2 Détails d'implémentation

Il ne vous est pas fourni de code source pour ce projet. Cependant, vous trouverez sur Madoc un jeu d'images au format PGM de tailles variables, `images.tar.gz`, permettant de tester vos programmes sur de petites ou grandes images. Pour lire et écrire dans un système de fichiers, vous pourrez utiliser la libraire standard `java.io.File` ; les fichiers à générer étant des fichiers en texte brut, vous n'aurez que des `String` à manipuler.

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

1 — un constructeur, prenant en paramètre un chemin vers un fichier PGM et construisant le `Quadtree` correspondant ;

2 — une méthode `toString`, ne prenant pas de paramètre, et produisant la représentation textuelle du quadtree sous forme parenthésée (comme vu en TD2) où chaque valeur de luminosité sera représentée par sa valeur décimale ; par exemple, le *quadtree* donné dans la Figure 3 est représenté par

```
((0 1 3 2) (4 5 7 6) (8 9 11 10) (12 13 15 14)).
```

3 — une méthode `toPGM`, prenant en paramètre un chemin, et générant à cet endroit le fichier PGM correspondant au `Quadtree`.

4 — une méthode `compressLambda`, ne prenant pas de paramètre, et appliquant la première technique de compression (cf. Section 2.3.1) ;

5 — une méthode `compressRho`, prenant en paramètre un entier $0 \leq \rho \leq 100$ et appliquant la seconde technique de compression (cf. Section 2.3.2) ;

Ces méthodes pourront utiliser des fonctions et structures de données auxiliaires, ainsi que d'autres classes que vous aurez créées si cela permet un gain de performance ; le cas échéant, ce sera motivé dans le rapport.

Vous écrirez un programme principal qui permettra, via un menu textuel, d'accéder aux fonctionnalités suivantes et les enchaîner de toute manière logique possible [3] :

main

— Choisir parmi une liste d'images PGM une image à charger ;
— Appliquer une compression Lambda à cette image et sauvegarder le *quadtree* resultant dans un fichier ;
— Appliquer une compression Rho pour une valeur ρ à entrer et sauvegarder le *quadtree* resultant dans un fichier ;
— Générer le fichier PGM correspondant au *quadtree* précédemment construit.
— Afficher les statistiques de compression : le nombre de noeuds du *quadtree* initial, le nombre de noeuds du *quadtree* résultant après la compression choisie (Lambda ou Rho), et le taux de compression réalisé.

Le programme devra également pouvoir être exécuté en mode non-interactif en l'invoquant avec en paramètre le nom d'un fichier PGM et un entier ρ. Dans ce cas, il appliquera automatiquement chacune des deux méthodes de compression sur le fichier brut et générera les fichiers correspondant (contenant les *quadtrees* et les nouvelles images), en affichant les statistiques de compression pour chacune des compressions appliquées.

3.3 Démonstration

Votre programme devra être opérationnel pour votre dernière séance de TP encadré sur le projet, en semaine 49 (04/12 - 08/12) ; votre chargé·e de TP passera vérifier son fonctionnement au cours d'une courte démonstration (environ 5 minutes) durant cette séance.

3. On pourra choisir, par exemple, de demander une image à charger, puis - en revenant au menu - d'appliquer une compression Rho, puis - en revenant au menu - de voir le quadtree résultant, puis d'appliquer une compression Lambda, etc.

Références

[1] SANDVINE. Phenomena : The global internet phenomena report. Tech. rep., Sandvine Corporation, Jan. 2023.