

**TUGAS KECL 2**  
**IF2211 - STRATEGI ALGORITMA**  
**KOMPRESI GAMBAR DENGAN**  
**QUADTREE**



Oleh:

David Bakti Lodianto	13523083
Lutfi Hakim Yusra	13523084

**PROGRAM STUDI TEKNIK INFORMATIKA**  
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**JL. GANESA 10, BANDUNG 40132**  
**2025**

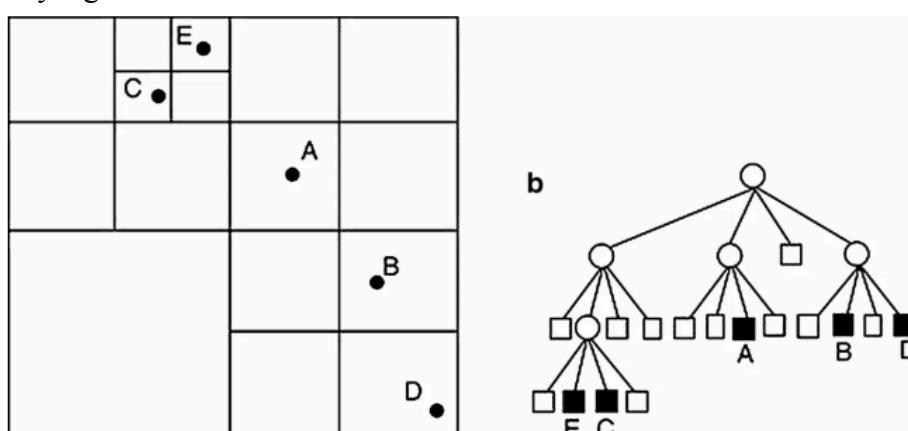
# DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I: ALGORITMA</b>	<b>3</b>
<b>BAB II: IMPLEMENTASI</b>	<b>4</b>
<b>BAB III: PENGUJIAN</b>	<b>5</b>
<b>BAB IV: ANALISIS</b>	<b>6</b>
<b>BAB V: BONUS</b>	<b>7</b>
<b>LAMPIRAN</b>	<b>8</b>

# BAB I: ALGORITMA

## 1. Quadtree

*Quadtree* adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, *Quadtree* membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.



Gambar 1 Visualisasi *Quadtree* dalam Kompresi Gambar

Sumber: <https://medium.com/@tannerwyrk/quadtrees-for-image-processing-302536c95c00>

Dalam implementasi teknis, sebuah *Quadtree* direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. *Quadtree* sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

## 2. Algoritma Divide and Conquer

Penggunaan struktur data *Quadtree* memerlukan algoritma divide and conquer, yang membagi *Quadtree* ke 4 simpul anak secara terus menerus hingga node tersebut mencapai keseragaman pada pixel-pixel dalamnya. Pembuatan suatu *Quadtree* mengikuti algoritma sebagai berikut.

- a. Definisikan node *root* dari *Quadtree*, yang menyimpan nilai posisi titik ujung kiri atas gambar (0,0) serta panjang dan lebar gambar, dan kedalaman 0.

- b. Uji keseragaman node. Jika dianggap tidak seragam dan ukurannya belum cukup kecil, ***divide***:
  - i. Definisikan 4 node yang membagi node *parent*-nya menjadi 4 bagian sama besar, dengan masing-masing menyimpan nilai posisi titik ujung kiri dan serta panjang dan lebar tiap bagian gambarnya, dan kedalamannya yakni kedalaman node *parent* ditambahkan satu.
  - ii. Ulangi langkah (b) untuk setiap node-nya.
- c. Setelah setiap node sudah cukup kecil atau cukup seragam, ambil kedalaman maksimum dari masing-masing *child node* dari node tersebut dan ambil nilai kedalaman terbesar dari keempatnya. Kemudian, jadikan nilainya sebagai kedalaman maksimum node tersebut. Selain itu, simpan nilai rata-rata warna tiap pixel dalam posisi dan ukuran *node*.

Setelah *Quadtree* berhasil dikonstruksi, langkah selanjutnya adalah menghasilkan gambar terkompresi berdasarkan struktur data ini, dengan algoritma sebagai berikut.

- a. Mulai dari node *root* dari *Quadtree*, serta kedalaman yang diinginkan.
  - i. Jika *node* merupakan *leaf node* atau kedalamannya nol, hasilkan gambar menurut ukurannya dengan satu warna seragam yang merupakan rata-rata dari semua warna pixel dalam *node* tersebut.
  - ii. Jika tidak, buat dan hasilkan 4 gambar dari tiap *child node*, lalu gabungkan menjadi satu gambar. Pembuatan gambar pada *child node* dapat dimulai dari langkah (i).

Dengan dijalankannya algoritma ini, gambar dapat dikompresi dengan metode quadtree.

## BAB II: IMPLEMENTASI

Pemrograman aplikasi kompresi gambar ini menggunakan bahasa C++, sehingga menggunakan paradigma Object-Oriented Programming (dengan sedikit paradigma prosedural karena external library yang tidak mendukung OOP).

Aplikasi ini menggunakan sejumlah 3 external library, sebagai berikut.

### 1. **stb\_image**

Library pemrosesan gambar, khususnya pada *read* dan *writing*, dalam bahasa C.

### 2. **eigen**

Library aljabar linier, digunakan untuk pemrosesan matriks.

### 3. **giflib**

Library pemrosesan gambar bertipe gif.

Dalam implementasi, aplikasi ini menggunakan 5 kelas, dan 1 program utama.

### 1. Class Image

#### o Attribute

Nama	Tipe	Deskripsi
width, height	private int	Nilai lebar dan tinggi dari gambar
red, green, blue	private MatrixXd	Matriks double (0 - 1) untuk menyimpan nilai red, green, blue di tiap pixel.

```
private:  
    int width, height;  
    MatrixXd red, green, blue;
```

#### o Method

Nama	Tipe	Parameter	Deskripsi
Image	Constructor		Default constructor dari kelas Image, width dan height 0, matriks warna kosong.
		<pre>Image::Image() : width(0), height(0) {}</pre>	
Image	Constructor	const char* filename	Constructor dari kelas Image,

			mengisi atribut-atributnya dengan gambar pada path.
		<pre>Image::Image(const char* filename) {     loadImage(filename); }</pre>	
Image	Constructor	const char* filename	Constructor dari kelas Image, mengisi atribut-atributnya dengan gambar pada path.
		<pre>Image::Image(const char* filename) {     loadImage(filename); }</pre>	
Image	Constructor	const Image& topLeft, const Image& topRight, const Image& bottomLeft, const Image& bottomRight	Constructor dari kelas Image, menggabungkan 4 gambar menjadi satu.
		<pre>Image::Image(const char* filename) {     loadImage(filename); }</pre>	
Image	Constructor	const char* filename	Constructor dari kelas Image, mengisi atribut-atributnya dengan gambar pada path.

```

Image::Image(const Image& topLeft, const Image& topRight, const Image& bottomLeft,
             const Image& bottomRight) {
    width = topLeft.width + topRight.width;
    height = topLeft.height + bottomLeft.height;

    red.resize(height, width);
    green.resize(height, width);
    blue.resize(height, width);

    red.block(0, 0, topLeft.height, topLeft.width) = topLeft.red;
    red.block(0, topLeft.width, topRight.height,
              topRight.width) = topRight.red;

    red.block(topLeft.height, 0, bottomLeft.height,
              bottomLeft.width) = bottomLeft.red;
    red.block(topLeft.height, topLeft.width, bottomRight.height,
              bottomRight.width) = bottomRight.red;

    green.block(0, 0, topLeft.height, topLeft.width) = topLeft.green;
    green.block(0, topLeft.width, topRight.height,
               topRight.width) = topRight.green;

    green.block(topLeft.height, 0, bottomLeft.height,
               bottomLeft.width) = bottomLeft.green;
    green.block(topLeft.height, topLeft.width, bottomRight.height,
               bottomRight.width) = bottomRight.green;

    blue.block(0, 0, topLeft.height, topLeft.width) = topLeft.blue;
    blue.block(0, topLeft.width, topRight.height,
              topRight.width) = topRight.blue;

    blue.block(topLeft.height, 0, bottomLeft.height,
               bottomLeft.width) = bottomLeft.blue;
    blue.block(topLeft.height, topLeft.width, bottomRight.height,
               bottomRight.width) = bottomRight.blue;
}

```

loadImage	void	const char* filename	Memuat gambar pada path ke dirinya.
-----------	------	----------------------	-------------------------------------

```

void Image::loadImage(const char* inputPath) {
    unsigned char* img = stbi_load(inputPath, &width, &height, nullptr, 3);
    if (img == nullptr) {
        cerr << "Failed to load image: " << inputPath << endl;
        return;
    }

    red.resize(height, width);
    green.resize(height, width);
    blue.resize(height, width);

    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            red(i, j) = img[(i * width + j) * 3] / 255.0;
            green(i, j) = img[(i * width + j) * 3 + 1] / 255.0;
            blue(i, j) = img[(i * width + j) * 3 + 2] / 255.0;
        }
    }
    stbi_image_free(img);
}

```

saveImage	void const	const char* outputPath	Menyimpan gambar pada path.
<pre>void Image::saveImage(const char* outputPath) const {     int channels = 3;     unsigned char* img = new unsigned char[width * height * channels];      for (int i = 0; i &lt; height; ++i) {         for (int j = 0; j &lt; width; ++j) {             img[(i * width + j) * channels] =                 static_cast&lt;unsigned char&gt;(red(i, j) * 255);              img[(i * width + j) * channels + 1] =                 static_cast&lt;unsigned char&gt;(green(i, j) * 255);              img[(i * width + j) * channels + 2] =                 static_cast&lt;unsigned char&gt;(blue(i, j) * 255);         }     }      if (!stbi_write_png(outputPath, width, height, channels, img,                         width * channels)) {         cerr &lt;&lt; "Failed to save image: " &lt;&lt; outputPath &lt;&lt; endl;     }      delete[] img; }</pre>			
printImageDetails	void		Mencetak detail gambar, untuk debugging.
<pre>void Image::printImageDetails() {     cout &lt;&lt; "Image Details:" &lt;&lt; endl;     cout &lt;&lt; "Width: " &lt;&lt; width &lt;&lt; ", Height: " &lt;&lt; height &lt;&lt; endl; }</pre>			
<b>Getters</b>			
getWidth getHeight	double const		Mengembalikan nilai konstan ukuran gambar.
<pre>int getWidth() const { return width; } int getHeight() const { return height; }</pre>			
Red Green Blue	MatrixXd const		Mengembalikan matriks konstan

```

MatrixXd Red() const { return red; }
MatrixXd Green() const { return green; }
MatrixXd Blue() const { return blue; }

```

redAt greenAt blueAt	double const	int x, int y	Mengembalikan nilai warna dari matriks warna pada posisi (x, y)
----------------------------	--------------	--------------	---

```

double redAt(int x, int y) const { return red(y, x); }
double greenAt(int x, int y) const { return green(y, x); }
double blueAt(int x, int y) const { return blue(y, x); }

```

getAvgRedBlock getAvgGreenBlock getAvgBlueBlock	double const	int x, int y, int width, int height	Mengembalikan nilai rerata warna blok.
---	--------------	---	--

```

double getAvgRedBlock(int x, int y, int width, int height) const {
    return red.block(y, x, height, width).mean(); }

double getAvgGreenBlock(int x, int y, int width, int height) const {
    return green.block(y, x, height, width).mean(); }

double getAvgBlueBlock(int x, int y, int width, int height) const {
    return blue.block(y, x, height, width).mean(); }

```

#### Setters

setColorRed setColorGreen setColorBlue	void	double avg	Mengubah semua nilai pada matriks warna ke suatu nilai konstan.
--	------	------------	---

```

void setColorRed(double avg) { red.setConstant(avg); }
void setColorGreen(double avg) { green.setConstant(avg); }
void setColorBlue(double avg) { blue.setConstant(avg); }

```

```
#ifndef IMAGE_HPP
#define IMAGE_HPP

#include <Eigen/Dense>
#include <stb_image.h>
#include <stb_image_write.h>
#include <iostream>
#include <string>

using namespace Eigen;
using namespace std;

class Image {
private:
    int width, height;
    MatrixXd red, green, blue;

public:
    Image();
    Image(const char* filename);
    Image(const int width, const int height);
    Image(const Image& topLeft, const Image& topRight,
          const Image& bottomLeft, const Image& bottomRight);

    void loadImage(const char* filename);
    void saveImage(const char* filename) const;

    int getWidth() const { return width; }
    int getHeight() const { return height; }

    MatrixXd Red() const { return red; }
    MatrixXd Green() const { return green; }
    MatrixXd Blue() const { return blue; }

    double redAt(int x, int y) const { return red(y, x); }
    double greenAt(int x, int y) const { return green(y, x); }
    double blueAt(int x, int y) const { return blue(y, x); }

    void setColorRed(double avg) { red.setConstant(avg); }
    void setColorGreen(double avg) { green.setConstant(avg); }
    void setColorBlue(double avg) { blue.setConstant(avg); }

    double getAvgRedBlock(int x, int y, int width, int height) const {
        return red.block(y, x, height, width).mean();
    }

    double getAvgGreenBlock(int x, int y, int width, int height) const {
        return green.block(y, x, height, width).mean();
    }

    double getAvgBlueBlock(int x, int y, int width, int height) const {
        return blue.block(y, x, height, width).mean();
    }

    void printImageDetails();
};

#endif
```

## 2. Class Quadtree

- **Attribute**

Nama	Tipe	Deskripsi
x, y	private int	Koordinat pojok kiri atas dari balok
width, height	private int	Nilai panjang dan lebar dari balok
isLeaf	private bool	Menentukan apakah quadTree memiliki anak atau tidak. False jika iya.
topLeftTree	pointer to QuadTree	Pointer menuju anak yang merepresentasikan potongan kiri atas dari foto
topRightTree	pointer to Quadtree	Pointer menuju anak yang merepresentasikan potongan kanan atas dari foto
bottomLeftTree	pointer to Quadtree	Pointer menuju anak yang merepresentasikan potongan bawah kiri dari foto
bottomRightTree	pointer to Quadtree	Pointer menuju anak yang merepresentasikan potongan bawah kanan dari foto
image	pointer to Image	Pointer menuju Image yang akan dikompresi
redVal, greenVal, blueVal	private double	Nilai rata-rata warna dari balok ini
maxDepth	private int	Kedalaman quadTree

```

private:
    int x, y; // Coordinates of the top-left corner of the block

    int width, height; // Dimensions of the block

    bool isLeaf; // True if the node is a leaf node, false if it has children

    QuadTree *topLeftTree; // Pointer to the top-left child
    QuadTree *topRightTree; // Pointer to the top-right child
    QuadTree *bottomLeftTree; // Pointer to the bottom-left child
    QuadTree *bottomRightTree; // Pointer to the bottom-right child

    Image *image; // Pointer to the image object

    double redVal, greenVal, blueVal; // Average color values for the block

    int maxDepth = 0;

```

- **Method**

Nama	Tipe	Parameter	Deskripsi
QuadTree	Constructor	Image pointer	Membangun sebuah quadtree tidak beranak dengan tumpuan panjang dan lebar dari image, dan juga pointer menuju image
		<pre> QuadTree::QuadTree(Image *img) : x(0), y(0), width(img-&gt;getWidth()), height(img-&gt;getHeight()), isLeaf(true), topLeftTree(nullptr),   topRightTree(nullptr), bottomLeftTree(nullptr), bottomRightTree(nullptr), image(img), maxDepth(0) {}</pre>	
QuadTree	Constructor	int x, int y, int width, int height, Image *img, int currentDepth	Menyusun quadtree dengan memasuki value secara manual untuk keseluruhan atribut
		<pre> QuadTree::QuadTree(int x, int y, int width, int height, Image *img, int currentDepth) : x(x), y(y), width(width), height(height), isLeaf(true), topLeftTree(nullptr),   topRightTree(nullptr), bottomLeftTree(nullptr), bottomRightTree(nullptr), image(img), maxDepth(currentDepth) {}</pre>	
QuadTree	Destructor	-	Melakukan delete terhadap objek quadTree

```

QuadTree::~QuadTree() {
    delete topLeftTree;
    topLeftTree = nullptr;
    delete topRightTree;
    topRightTree = nullptr;
    delete bottomLeftTree;
    bottomLeftTree = nullptr;
    delete bottomRightTree;
    bottomRightTree = nullptr;
}

```

divide	void	int currentDepth	Membagikan balok yang direpresentasikan node menjadi empat bagian, kiri atas, kanan atas, kiri bawah, kanan bawah, lalu memasukkannya semua sebagai anak dari node tersebut
--------	------	------------------	---

```

void QuadTree::divide(int currentDepth) {
    int halfWidth = width / 2;
    int halfHeight = height / 2;
    isLeaf = false;
    topLeftTree = new QuadTree(x, y, halfWidth, halfHeight, image, currentDepth);
    topRightTree = new QuadTree(x + halfWidth, y, width - halfWidth, halfHeight, image, currentDepth);
    bottomLeftTree = new QuadTree(x, y + halfHeight, halfWidth, height - halfHeight, image,
        currentDepth);
    bottomRightTree = new QuadTree(x + halfWidth, y + halfHeight, width - halfWidth, height -
        halfHeight, image, currentDepth);
}

```

buildTree	void	int minBlockSize, double thresholdValue, double (*errorMeasure)(c onst Image&, int, int, int, int))	Untuk sebuah node, melakukan uji keseragaman dengan error measure yang ditentukan dan nilai threshold yang diharapkan. Jika belum memenuhi syarat, memotongnya jadi empat, dan hal yang sama terhadap masing-masing. Akhirnya, bahanukan nilai maxDepth
-----------	------	---	---

```

void QuadTree::buildTree(int minBlockSize, double thresholdValue, double (*errorMeasure)(const Image&, int, int, int)) {
    double error = errorMeasure(*image, x, y, x + width - 1, y + height - 1);
    if (width <= 1 || height <= 1 || width * height <= minBlockSize || error < thresholdValue) {
        isLeaf = true;
    } else {
        divide(maxDepth + 1);
        topLeftTree->buildTree(minBlockSize, thresholdValue, errorMeasure);
        topRightTree->buildTree(minBlockSize, thresholdValue, errorMeasure);
        bottomLeftTree->buildTree(minBlockSize, thresholdValue, errorMeasure);
        bottomRightTree->buildTree(minBlockSize, thresholdValue, errorMeasure);
        maxDepth = max({topLeftTree->maxDepth, topRightTree->maxDepth, bottomLeftTree->maxDepth,
                        bottomRightTree->maxDepth});
    }
    setAverageColors();
}

```

setAverageColors	void	-	Mengecek warna rata-rata dari detail balok yang tersedia pada node quadTree
------------------	------	---	---

```

void QuadTree::setAverageColors() {
    if (isLeaf) {
        redVal = image->getAvgRedBlock(x, y, width, height);
        greenVal = image->getAvgGreenBlock(x, y, width, height);
        blueVal = image->getAvgBlueBlock(x, y, width, height);
    } else {
        redVal = (topLeftTree->redVal + topRightTree->redVal + bottomLeftTree->redVal +
                  bottomRightTree->redVal) / 4.0;
        greenVal = (topLeftTree->greenVal + topRightTree->greenVal + bottomLeftTree->greenVal +
                    bottomRightTree->greenVal) / 4.0;
        blueVal = (topLeftTree->blueVal + topRightTree->blueVal + bottomLeftTree->blueVal +
                   bottomRightTree->blueVal) / 4.0;
    }
}

```

renderImage	Image	int depth	Melakukan konversi dari quadTree menjadi Image object dengan menggabungkan semua leaf atau yang di ujung depth. Gambar paling kecil merupakan nilai rata-rata warna balok tersebut.
-------------	-------	-----------	---

```

Image QuadTree::renderImage(int depth) {
    if (isLeaf || depth == 0) {
        Image renderedImage(width, height);
        renderedImage.setColorRed(redVal);
        renderedImage.setColorGreen(greenVal);
        renderedImage.setColorBlue(blueVal);
        return renderedImage;
    } else {
        Image topLeftImage = topLeftTree->renderImage(depth - 1);
        Image topRightImage = topRightTree->renderImage(depth - 1);
        Image bottomLeftImage = bottomLeftTree->renderImage(depth - 1);
        Image bottomRightImage = bottomRightTree->renderImage(depth - 1);
        Image combinedImage(topLeftImage, topRightImage, bottomLeftImage, bottomRightImage);
        return combinedImage;
    }
}

```

printNodeInfo	void	int depth	Untuk melakukan
---------------	------	-----------	-----------------

			debugging, memperlihatkan semua node.
<pre>void QuadTree::printNodeInfo(int depth) const {     cout &lt;&lt; "Node at (" &lt;&lt; x &lt;&lt; ", " &lt;&lt; y &lt;&lt; ") with size (" &lt;&lt; width &lt;&lt; ", " &lt;&lt; height &lt;&lt; ") - Depth: " &lt;&lt; depth &lt;&lt; endl;     cout &lt;&lt; "Average Colors: (" &lt;&lt; redVal &lt;&lt; ", " &lt;&lt; greenVal &lt;&lt; ", " &lt;&lt; blueVal &lt;&lt; ")" &lt;&lt; endl;     if (!isLeaf) {         if (topLeftTree) topLeftTree-&gt;printNodeInfo(depth + 1);         if (topRightTree) topRightTree-&gt;printNodeInfo(depth + 1);         if (bottomLeftTree) bottomLeftTree-&gt;printNodeInfo(depth + 1);         if (bottomRightTree) bottomRightTree-&gt;printNodeInfo(depth + 1);     }     cout &lt;&lt; "-----" &lt;&lt; endl; }</pre>			
getNodeCount	int	const char* filename	Mengembalikan jumlah node pada tree.
<pre>int QuadTree::getNodeCount() {     if (isLeaf) {         return 1;     } else {         return 1 + topLeftTree-&gt;getNodeCount() + topRightTree-&gt;getNodeCount() + bottomLeftTree-&gt;getNodeCount() + bottomRightTree-&gt;getNodeCount();     } }</pre>			
binarySearchThreshold	double	const char* inputPath, const char* outputPath, int minBlockSize, Image& image, double (*errorMeasure)(const Image&, int, int, int, int), double low, double high, double target compressionRatio, int iterations = 10))	Melakukan binary search pada threshold, dengan menggunakan nilai tengah dari minimum dan maximum yang mungkin dari metode error untuk mendapatkan threshold yang memberikan rasio kompresi yang sesuai. Jika rasio terlalu besar, nilai tengah menjadi minimum, dan ulangi. Jika terlalu kecil, nilai tengah menjadi maximum. Kembalikan nilai threshold jika dekat atau setelah sepuluh iterasi

```
double QuadTree::binarySearchThreshold(const char* inputPath, const char* outputPath, int minBlockSize,
Image& image, double (*errorMeasure)(const Image&, int, int, int, int), double low, double high, double
targetCompressionRatio, int iterations) {
    double thresholdTolerance = 0.01;

    if(iterations <= 0) {
        return (low + high) / 2.0;
    }

    double midThreshold = (low + high) / 2.0;
    QuadTree quadTree(&image);
    quadTree.buildTree(minBlockSize, midThreshold, errorMeasure);
    Image renderedImage = quadTree.renderImage(quadTree.getMaxDepth());
    renderedImage.saveImage(outputPath);
    double originalSize = ErrorMeasure::getFileSize(inputPath);
    double compressedSize = ErrorMeasure::getFileSize(outputPath);

    double currentCompressionRatio = compressedSize/originalSize;

    if (abs(currentCompressionRatio - targetCompressionRatio) < thresholdTolerance) {
        return midThreshold;
    } else if (currentCompressionRatio < targetCompressionRatio) {
        return binarySearchThreshold(inputPath, outputPath, minBlockSize, image, errorMeasure, low,
midThreshold, targetCompressionRatio, iterations - 1);
    } else {
        return binarySearchThreshold(inputPath, outputPath, minBlockSize, image, errorMeasure,
midThreshold, high, targetCompressionRatio, iterations - 1);
    }
}
```

```
● ● ●

#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include "ErrorMeasure.hpp"

using namespace std;

class QuadTree {
private:
    int x, y; // Coordinates of the top-left corner of the block
    int width, height; // Dimensions of the block
    bool isLeaf; // True if the node is a leaf node, false if it has children
    QuadTree *topLeftTree; // Pointer to the top-left child
    QuadTree *topRightTree; // Pointer to the top-right child
    QuadTree *bottomLeftTree; // Pointer to the bottom-left child
    QuadTree *bottomRightTree; // Pointer to the bottom-right child
    Image *image; // Pointer to the image object
    double redVal, greenVal, blueVal; // Average color values for the block
    int maxDepth = 0;

public:
    QuadTree(Image *image);
    QuadTree(int x, int y, int width, int height, Image *image, int currentDepth = 0);
    ~QuadTree();
    void divide(int currentDepth);
    void buildTree(int minBlockSize, double thresholdValue, double (*errorMeasure)(const Image&, int, int, int, int));
    void setAverageColors();
    int getX() const { return x; }
    int getY() const { return y; }
    int getWidth() const { return width; }
    int getHeight() const { return height; }
    bool isLeafNode() const { return isLeaf; }
    int getMaxDepth() const { return maxDepth; }
    QuadTree* getTopLeftTree() const { return topLeftTree; }
    QuadTree* getTopRightTree() const { return topRightTree; }
    QuadTree* getBottomLeftTree() const { return bottomLeftTree; }
    QuadTree* getBottomRightTree() const { return bottomRightTree; }
    Image renderImage(int depth);
    int getNodeCount();
    static double binarySearchThreshold(const char* inputFile, const char* outputFile, int minBlockSize, Image& image, double (*errorMeasure)(const Image&, int, int, int, int), double low, double high, double targetCompressionRatio, int iterations = 10);
    void printNodeInfo(int depth) const;
};

#endif // QUADTREE_HPP
```

### 3. Class Bucket

- **Attribute**

Nama	Tipe	Deskripsi
l, r	int	Nilai lebar dan tinggi dari gambar
colors	pointer to vector<tuple<double, double, double>>	vector yang menyimpan semua warna dari suatu gambar
res	pointer to vector<tuple<double, double, double>>	Vector yang menyimpan warna-warna yang telah di-kuantisasi.
depth	int	Kedalaman bucket
maxDepth	int	Kedalaman maksimum bucket.

```
int l, r;
vector<tuple<double, double, double>> *colors;
vector<tuple<double, double, double>> *res;
int depth;
int maxDepth;
```

- **Method**

Nama	Tipe	Parameter	Deskripsi
Bucket	Constructor	vector<tuple<double, double, double>> *colors, vector<tuple<double, double, double>> *res, int l, int r, int maxDepth, int depth = 0	Konstruktor kelas Bucket, sekaligus memulai proses kuantisasi warna dengan divide.

```
Bucket(vector<tuple<double, double, double>> *colors,
       vector<tuple<double, double, double>> *res,
       int l, int r, int maxDepth, int depth = 0)
: colors(colors), res(res), maxDepth(maxDepth),
  l(l), r(r), depth(depth) {
    divide(depth);
}
```

sortColors	void	int index	Mengurutkan warna pada vector colors sesuai index.
<pre> void sortColors(int index) {     if (index == 0) {         sort(colors-&gt;begin() + static_cast&lt;std::size_t&gt;(l),               colors-&gt;begin() + static_cast&lt;std::size_t&gt;(r),               [] (const tuple&lt;double, double, double&gt;&amp; a,                   const tuple&lt;double, double, double&gt;&amp; b) {                   return get&lt;0&gt;(a) &lt; get&lt;0&gt;(b);               });     } else if (index == 1) {         sort(colors-&gt;begin() + static_cast&lt;std::size_t&gt;(l),               colors-&gt;begin() + static_cast&lt;std::size_t&gt;(r),               [] (const tuple&lt;double, double, double&gt;&amp; a,                   const tuple&lt;double, double, double&gt;&amp; b) {                   return get&lt;1&gt;(a) &lt; get&lt;1&gt;(b);               });     } else if (index == 2) {         sort(colors-&gt;begin() + static_cast&lt;std::size_t&gt;(l),               colors-&gt;begin() + static_cast&lt;std::size_t&gt;(r),               [] (const tuple&lt;double, double, double&gt;&amp; a,                   const tuple&lt;double, double, double&gt;&amp; b) {                   return get&lt;2&gt;(a) &lt; get&lt;2&gt;(b);               });     } } </pre>			
divide	void	int depth	Membagi bucket menjadi dua, dengan melihat range kanal warna yang paling besar, diurutkan, dan dibagi di tengahnya.

```

void divide(int depth) {
    if (depth >= maxDepth) {
        double avgR = 0, avgG = 0, avgB = 0;
        for (int i = l; i <= r; i++) {
            avgR += get<0>((*colors)[i]);
            avgG += get<1>((*colors)[i]);
            avgB += get<2>((*colors)[i]);
        }
        avgR /= (r - l + 1);
        avgG /= (r - l + 1);
        avgB /= (r - l + 1);
        res->emplace_back(avgR, avgG, avgB);
        return;
    }
    int mid = (l + r) / 2;
    int index;
    double minR = 0, minG = 0, minB = 0;
    double maxR = 1, maxG = 1, maxB = 1;
    for (int i = l; i <= r; i++) {
        minR = min(minR, get<0>((*colors)[i]));
        maxR = max(maxR, get<0>((*colors)[i]));
        minG = min(minG, get<1>((*colors)[i]));
        maxG = max(maxG, get<1>((*colors)[i]));
        minB = min(minB, get<2>((*colors)[i]));
        maxB = max(maxB, get<2>((*colors)[i]));
    }
    double rangeR = maxR - minR;
    double rangeG = maxG - minG;
    double rangeB = maxB - minB;
    if (rangeR >= rangeG && rangeR >= rangeB) {
        index = 0;
    } else if (rangeG >= rangeR && rangeG >= rangeB) {
        index = 1;
    } else {
        index = 2;
    }
    sortColors(index);
    Bucket left(colors, res, l, mid, maxDepth, depth + 1);
    Bucket right(colors, res, mid + 1, r, maxDepth, depth + 1);
}

```

quantize	static vector<tuple<double, double, double>> >>	vector<tuple<double, double, double>> *colors, int colorCount	Menghasilkan <i>limited color palette</i> dalam bentuk <i>vector of tuple</i> dari semua warna pada gambar, sebanyak perpangkatan 2 terbesar yang lebih kecil dari colorCount
----------	---	--	---

```

static vector<tuple<double, double, double>> quantize(
    vector<tuple<double, double, double>> *colors, int colorCount) {
    vector<tuple<double, double, double>> res;
    res.reserve(colorCount);
    Bucket bucket(colors, &res, 0, colors->size() - 1,
                  log2(colorCount));
    return res;
}

```

findNearestColor	static int	double r, double g, double b, GifColorType *res, int colorCount	Mengembalikan index pada color palette yang terdekat dengan warna yang ingin dikonversi
------------------	------------	---	--

```

static int findNearestColor(double r, double g, double b,
                           GifColorType *res, int colorCount) {
    int nearestIndex = -1;
    double minDistance = -1;
    for (int i = 0; i < colorCount; i++) {
        double distance = sqrt(pow(r - res[i].Red, 2) +
                               pow(g - res[i].Green, 2) +
                               pow(b - res[i].Blue, 2));

        if (nearestIndex == -1 || distance < minDistance) {
            minDistance = distance;
            nearestIndex = i;
        }
    }
    return nearestIndex;
}

```

createColorPalette	static GifColorType*	Image img, int colorCount	Mengembalikan <i>color palette</i> yang dapat langsung digunakan untuk GifFileType.
--------------------	-------------------------	------------------------------	---

```

GifColorType* GIF::createColorPalette(Image img, int colorCount) {
    vector<tuple<double, double, double>> colors;
    int width = img.getWidth();
    int height = img.getHeight();
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            colors.emplace_back(img.redAt(j, i), img.greenAt(j, i),
                                img.blueAt(j, i));
        }
    }
    vector<tuple<double, double, double>> palette =
        Bucket::quantize(&colors, colorCount);

    GifColorType* gifPalette = new GifColorType[colorCount];
    for (int i = 0; i < colorCount; i++) {
        gifPalette[i].Red   = static_cast<int>(get<0>(palette[i]) * 255);
        gifPalette[i].Green = static_cast<int>(get<1>(palette[i]) * 255);
        gifPalette[i].Blue  = static_cast<int>(get<2>(palette[i]) * 255);
    }

    return gifPalette;
}

```

saveGIF	static void	const char* fileName, Image* images, int frameCount	Menyimpan GIF proses kompresi pada path, berdasarkan array of Image sebagai frames.
---------	-------------	--	--

```

void GIF::saveGIF(const char* fileName, Image* images, int frameCount) {
    int colorCount = 256;
    GifColorType* palette = createColorPalette(images[frameCount - 1],
                                                colorCount);

    int width = images[0].getWidth();
    int height = images[0].getHeight();
    int error;
    GifFileType* gifFile = EGifOpenFileName(fileName, false, &error);
    if (!gifFile) {
        cout << "EGifOpenFileName() failed - " << error << endl;
    }

    gifFile->SWidth = width;
    gifFile->SHeight = height;
    gifFile->SColorResolution = 8;
    gifFile->SBackGroundColor = 0;
    gifFile->SColorMap = GifMakeMapObject(256, palette);

    unsigned char appData[] = "NETSCAPE2.0";
    GifAddExtensionBlock(&gifFile->ExtensionBlockCount,
                         &gifFile->ExtensionBlocks,
                         APPLICATION_EXT_FUNC_CODE,
                         11, appData);

    unsigned char loopData[3] = { 1, 0, 0 }; // Infinite loop
    GifAddExtensionBlock(&gifFile->ExtensionBlockCount,
                         &gifFile->ExtensionBlocks,
                         CONTINUE_EXT_FUNC_CODE,
                         3, loopData);

    for (int i = 0; i < frameCount; i++) {
        uint8_t* frameData = new uint8_t[width * height];

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int r = static_cast<int>(images[i].redAt(x, y) * 255);
                int g = static_cast<int>(images[i].greenAt(x, y) * 255);
                int b = static_cast<int>(images[i].blueAt(x, y) * 255);
                frameData[y * width + x] =
                    Bucket::findNearestColor(r, g, b, palette, colorCount);
            }
        }
        SavedImage frame;
        frame.ImageDesc.Left = 0;
        frame.ImageDesc.Top = 0;
        frame.ImageDesc.Width = width;
        frame.ImageDesc.Height = height;
        frame.ImageDesc.Interlace = false;
        frame.ImageDesc.ColorMap = nullptr;
        frame.RasterBits = frameData;
        frame.ExtensionBlockCount = 0;
        frame.ExtensionBlocks = nullptr;

        GifMakeSavedImage(gifFile, &frame);

        GraphicsControlBlock gcb;
        gcb.DisposalMode = DISPOSE_BACKGROUND;
        gcb.UserInputFlag = false;
        gcb.DelayTime = 100;
        gcb.TransparentColor = NO_TRANSPARENT_COLOR;

        EGifGCBToSavedExtension(&gcb, gifFile,
                               gifFile->ImageCount - 1);

        delete[] frameData;
    }

    if (EGifSpew(gifFile) == GIF_ERROR) {
        cout << "EGifSpew() failed - " << gifFile->Error << endl;
        EGifCloseFile(gifFile, &error);
    }
    error = 0;
}

```

```

#ifndef GIF_HPP
#define GIF_HPP

#include <gif_lib.h>
#include "Image.hpp"

class GIF {
public:
    static GifColorType* createColorPalette(Image img, int colorCount);
    static void saveGIF(const char* fileName, Image* images, int frameCount);
};

#endif

```

#### 4. Class ErrorMeasure

- Method

Nama	Tipe	Parameter	Deskripsi
Variance	static double	MatrixXd& matrix, int x1, int y1, int x2, int y2	Menghitung variance pada matriks sesuai dengan koordinat yang ditentukan. Variance merupakan jumlah dari kuadrat selisih tiap elemen terhadap mean, lalu dibagi jumlah elemen.
	<pre> double ErrorMeasure::Variance(const MatrixXd&amp; matrix, int x1, int y1, int x2, int y2) {     double mean = matrix.block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();     int rows = y2 - y1 + 1;     int cols = x2 - x1 + 1;     double variance = 0.0;      for (int i = y1; i &lt; y2 + 1; ++i) {         for (int j = x1; j &lt; x2 + 1; ++j) {             variance += (matrix(i, j) - mean) * (matrix(i, j) - mean);         }     }     return variance / (rows * cols); } </pre>		

```

double ErrorMeasure::varianceThreshold(const Image& image, int x1, int y1, int x2, int y2) {
    double redVar = Variance(image.Red(), x1, y1, x2, y2);
    double greenVar = Variance(image.Green(), x1, y1, x2, y2);
    double blueVar = Variance(image.Blue(), x1, y1, x2, y2);

    double avgVar = (redVar + greenVar + blueVar) / 3.0;

    return avgVar;
}

```

MeanAbsoluteDeviation	static double	const MatrixXd& matrix, int x1, int y1, int x2, int y2	Menghitung jumlah dari selisih tiap elemen terhadap mean, yang lalu dibagi jumlah elemen. Nilai tersebut merupakan MAD
-----------------------	---------------	--	--

```

double ErrorMeasure::MeanAbsoluteDeviation(const MatrixXd& matrix, int x1, int y1, int x2, int y2) {
    double mean = matrix.block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();
    int rows = y2 - y1 + 1;
    int cols = x2 - x1 + 1;
    double mad = 0.0;

    for (int i = y1; i < y2 + 1; ++i) {
        for (int j = x1; j < x2 + 1; ++j) {
            mad += abs(matrix(i, j) - mean);
        }
    }
    return mad / (rows * cols);
}

```

meanAbsoluteDeviationThreshold	static double	const Image& image, int x1, int y1, int x2, int y2	Menghitung MAD rata-rata dari image pada balok koordinat tertentu
--------------------------------	---------------	--	---

```

double ErrorMeasure::meanAbsoluteDeviationThreshold(const Image& image, int x1, int y1, int x2, int y2)
{
    double redMAD = MeanAbsoluteDeviation(image.Red(), x1, y1, x2, y2);
    double greenMAD = MeanAbsoluteDeviation(image.Green(), x1, y1, x2, y2);
    double blueMAD = MeanAbsoluteDeviation(image.Blue(), x1, y1, x2, y2);

    double avgMAD = (redMAD + greenMAD + blueMAD) / 3.0;

    return avgMAD;
}

```

MaxPixelDifference	static double	const MatrixXd& matrix, int x1, int y1, int x2, int y2	Menghitung nilai selisih terbesar yang mungkin antara dua nilai pada sebuah matrix pada koordinat balok tertentu
--------------------	---------------	--	--

```

double ErrorMeasure::MaxPixelDifference(const MatrixXd& matrix, int x1, int y1, int x2, int y2) {
    MatrixXd block = matrix.block(y1, x1, y2 - y1 + 1, x2 - x1 + 1);
    double min = block.minCoeff();
    double max = block.maxCoeff();

    return max - min;
}

```

maxPixelDifferenceThreshold	static double	const Image& image, int x1, int y1, int x2, int y2	Menghitung nilai selisih terbesar yang mungkin antara dua nilai rata-rata pada sebuah Image pada koordinat balok tertentu
<pre>double ErrorMeasure::maxPixelDifferenceThreshold(const Image&amp; image, int x1, int y1, int x2, int y2) {     double redMaxDiff = MaxPixelDifference(image.Red(), x1, y1, x2, y2);     double greenMaxDiff = MaxPixelDifference(image.Green(), x1, y1, x2, y2);     double blueMaxDiff = MaxPixelDifference(image.Blue(), x1, y1, x2, y2);      double avgMaxDiff = (redMaxDiff + greenMaxDiff + blueMaxDiff) / 3.0;      return avgMaxDiff; }</pre>			
entropyError	static double	const MatrixXd& matrix, int x1, int y1, int x2, int y2	Menghitung nilai entropy pada sebuah matriks pada koordinat balok yang ditentukan. Entropy merupakan nilai tingkat variasi sebuah matriks.
<pre>double ErrorMeasure::entropyError(const MatrixXd&amp; matrix, int x1, int y1, int x2, int y2) {     double entropy = 0.0;     double total = 0.0;     double histogram[256] = {0};      for (int i = y1; i &lt; y2 + 1; ++i) {         for (int j = x1; j &lt; x2 + 1; ++j) {             histogram[(int)(matrix(i, j) * 255)]++;             total++;         }     }      for (int i = 0; i &lt; 256; ++i) {         if (histogram[i] &gt; 0) {             double p = histogram[i] / total;             entropy -= p * log2(p);         }     }      return entropy; }</pre>			
entropyErrorThresh old	static double	const Image& image, int x1, int y1, int x2, int y2	Menghitung entropy rata-rata dari sebuah Image pada koordinat balok yang ditentukan

```

double ErrorMeasure::entropyErrorThreshold(const Image& image, int x1, int y1, int x2, int y2) {
    double redEntropy = entropyError(image.Red(), x1, y1, x2, y2);
    double greenEntropy = entropyError(image.Green(), x1, y1, x2, y2);
    double blueEntropy = entropyError(image.Blue(), x1, y1, x2, y2);

    double avgEntropy = (redEntropy + greenEntropy + blueEntropy) / 3.0;

    return avgEntropy;
}

```

getFileSize	double	const char* filename	Mengembalikan berat sebuah file dalam KB
-------------	--------	----------------------	--

```

double ErrorMeasure::getFileSize(const char* filename) {
    FILE *p_file = NULL;
    p_file = fopen(filename, "rb");
    if (!p_file) {
        cerr << "Error opening file" << endl;
        return -1;
    }
    fseek(p_file, 0, SEEK_END);
    int size = ftell(p_file);
    fclose(p_file);
    return double(size) / 1024.0;
}

```

SSIM Threshold	double	const Image& image, int x1, int y1, int x2, int y2.	Menghitung SSIM pada bagian image jika dibandingkan dengan flat average color dari sebuah balok yang ditentukan.
----------------	--------	---	--

```

double ErrorMeasure::SSIMThreshold(const Image& image1, int x1, int y1, int x2, int y2) {
    // original.loadImage("test/example.png");
    // C1 from 8-bit, normalized (0-1)

    Image image2(image1);
    double blue = image2.getAvgBlueBlock(x1, y1, x2 - x1 + 1, y2 - y1 + 1);
    double green = image2.getAvgGreenBlock(x1, y1, x2 - x1 + 1, y2 - y1 + 1);
    double red = image2.getAvgRedBlock(x1, y1, x2 - x1 + 1, y2 - y1 + 1);
    image2.setColorRed(red);
    image2.setColorGreen(green);
    image2.setColorBlue(blue);

    double C1 = pow(0.01, 2);

    // C2 from 8-bit, normalized (0-1)
    double C2 = pow(0.03, 2);

    // FOR CURRENT IMAGE
    double redMean = image1.Red().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();
    double greenMean = image1.Green().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();
    double blueMean = image1.Blue().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();

    double redVar = Variance(image1.Red(), x1, y1, x2, y2);
    double greenVar = Variance(image1.Green(), x1, y1, x2, y2);
    double blueVar = Variance(image1.Blue(), x1, y1, x2, y2);

    // FOR ORIGINAL IMAGE
    double originalRedMean = image2.Red().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();
    double originalGreenMean = image2.Green().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();
    double originalBlueMean = image2.Blue().block(y1, x1, y2 - y1 + 1, x2 - x1 + 1).mean();

    double originalRedVar = Variance(image2.Red(), x1, y1, x2, y2);
    double originalGreenVar = Variance(image2.Green(), x1, y1, x2, y2);
    double originalBlueVar = Variance(image2.Blue(), x1, y1, x2, y2);

    // SSIM
    double redCov = Covariance(image1.Red(), image2.Red(), x1, y1, x2, y2);
    double greenCov = Covariance(image1.Green(), image2.Green(), x1, y1, x2, y2);
    double blueCov = Covariance(image1.Blue(), image2.Blue(), x1, y1, x2, y2);

    double redSSIM = (2 * redMean * originalRedMean + C1) * (2 * redCov + C2) / ((pow(redMean, 2) +
    pow(originalRedMean, 2) + C1) * (redVar + originalRedVar + C2));
    double greenSSIM = (2 * greenMean * originalGreenMean + C1) * (2 * greenCov + C2) /
    ((pow(greenMean, 2) + pow(originalGreenMean, 2) + C1) * (greenVar + originalGreenVar + C2));
    double blueSSIM = (2 * blueMean * originalBlueMean + C1) * (2 * blueCov + C2) / ((pow(blueMean, 2) +
    pow(originalBlueMean, 2) + C1) * (blueVar + originalBlueVar + C2));

    double avgSSIM = (redSSIM + greenSSIM + blueSSIM) / 3.0;

    return -avgSSIM;
}

```

```
#ifndef ERROR_MEASURE_HPP
#define ERROR_MEASURE_HPP

#include "Image.hpp"
#include <math.h>
#include <fstream>

using namespace Eigen;
using namespace std;

class ErrorMeasure {
public:
    static double Variance(const MatrixXd& matrix, int x1, int y1, int x2, int y2);
    static double Covariance(const MatrixXd& matrix, const MatrixXd& matrix2, int x1, int y1, int x2,
int y2);
    static double MeanAbsoluteDeviation(const MatrixXd& matrix, int x1, int y1, int x2, int y2);
    static double MaxPixelDifference(const MatrixXd& matrix, int x1, int y1, int x2, int y2);
    static double entropyError(const MatrixXd& matrix, int x1, int y1, int x2, int y2);

    static double varianceThreshold(const Image& image, int x1, int y1, int x2, int y2); //0.0-0.25
    static double meanAbsoluteDeviationThreshold(const Image& image, int x1, int y1, int x2, int
y2); //0.0-0.5
    static double maxPixelDifferenceThreshold(const Image& image, int x1, int y1, int x2, int
y2); //0.0-1.0
    static double entropyErrorThreshold(const Image& image, int x1, int y1, int x2, int y2); //0.0-8.0
    static double SSIMThreshold(const Image& image1, const Image& image2, int x1, int y1, int x2, int
y2); //0.0-1.0

    static double getFileSize(const char* filename);
};

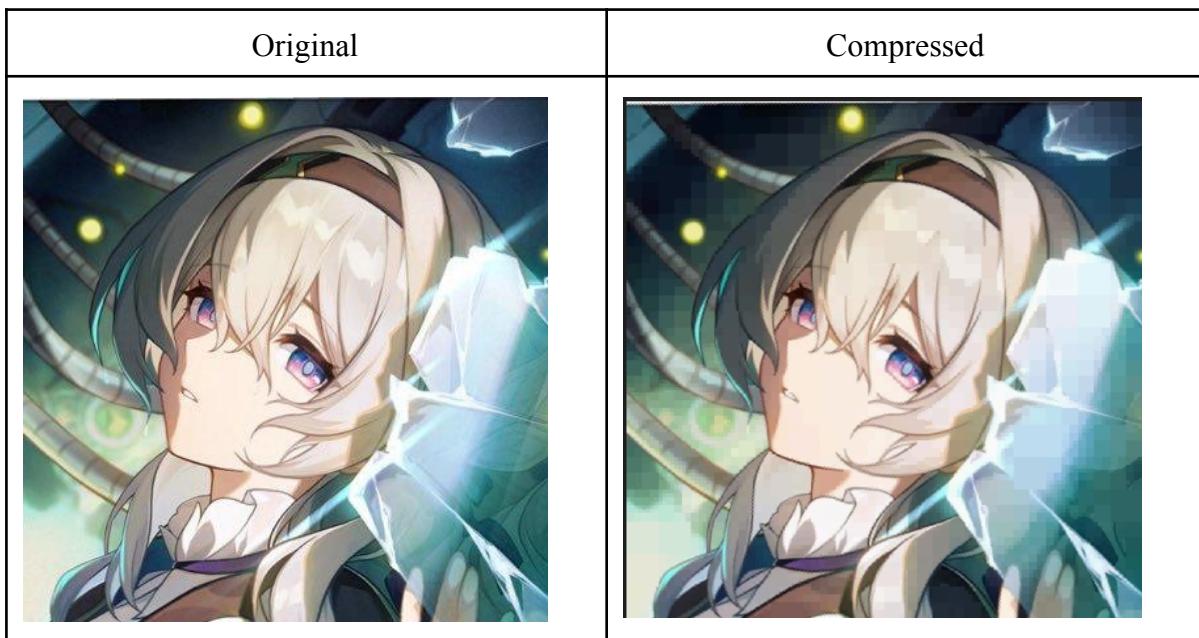
#endif
```

## BAB III: PENGUJIAN

### 1. firefly.jpg (Variance Threshold)

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly.jpg
-> Error Method Choice: 1
-> Threshold Value: 0.001
-> Target Compression Ratio: 0
-> Minimum Block Size: 1
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed.jpg
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed.gif
=====

=====
Image saved successfully.
=====
Image compression completed in 33933 milliseconds.
GIF creation completed in 49716 milliseconds.
Previous image size: 35.0068 KB
Compressed image size: 168.746 KB
Compression ratio: 482.038%
Depth of quadtree: 9
Number of nodes in quadtree: 49065
Compressed image saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed.jpg
GIF saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed.gif
=====
```



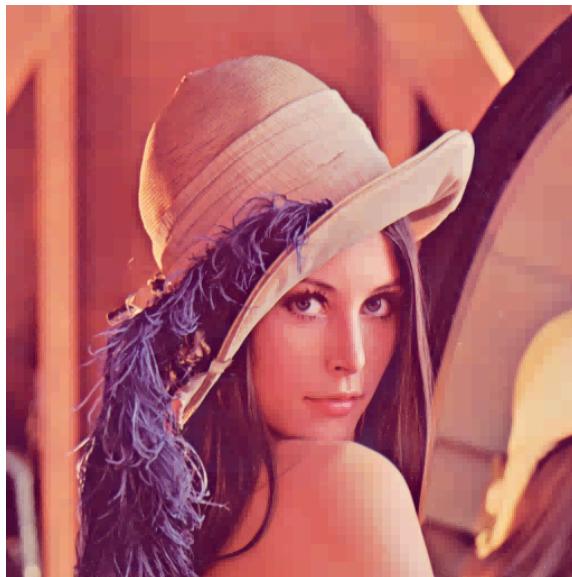
### 2. lena.png (Max Pixel Difference, Target Percentage Compression)

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\example.png
-> Error Method Choice: 3
-> Threshold Value: 0.1
-> Target Compression Ratio: 0.41
-> Minimum Block Size: 1
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\output.png
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\output.gif
=====

Are you sure?

Press ENTER to continue or CTRL+C to cancel...
```

```
=====
compressing...
=====
Image saved successfully.
=====
Image compression completed in 349586 milliseconds.
GIF creation completed in 350962 milliseconds.
Previous image size: 462.726 KB
Compressed image size: 192.424 KB
Compression ratio: 41.5849%
Depth of quadtree: 9
Number of nodes in quadtree: 63993
Compressed image saved to: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\lena_compressed.png
GIF saved to: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\lena_compressed.gif
=====
```

Original	Compressed
	

### 3. miyabi.jpg (Max Pixel Difference Threshold)

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\miyabi.png
-> Error Method Choice: 3
-> Threshold Value: 0.1
-> Target Compression Ratio: 0
-> Minimum Block Size: 1
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\miyabi_compressed.png
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\miyabi_compressed.gif
=====
Are you sure?
```

```
=====
Image saved successfully.
=====
Image compression completed in 47011 milliseconds.
GIF creation completed in 68728 milliseconds.
Previous image size: 252.991 KB
Compressed image size: 131.582 KB
Compression ratio: 52.0105%
Depth of quadtree: 9
Number of nodes in quadtree: 53909
Compressed image saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\miyabi_compressed.png
GIF saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\miyabi_compressed.gif
=====
```

Original	Compressed
----------	------------



#### 4. vergil.jpg (Mean Absolute Deviation Threshold)

```
=====
ViltrumCompressor v4.2
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\vergil.jpg
-> Error Method Choice: 2
-> Threshold Value: 0.1
-> Target Compression Ratio: 0
-> Minimum Block Size: 64
-> Absolute Path to Output Image: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\vergil_compressed.jpg
-> Absolute Path to Output GIF: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\vergil_compressed.gif
=====
Are you sure?
Press ENTER to continue or CTRL+C to cancel...
=====

compressing...
=====
Image saved successfully.
=====
Image compression completed in 502 milliseconds.
GIF creation completed in 1818 milliseconds.
Previous image size: 29.5342 KB
Compressed image size: 39.1455 KB
Compression ratio: 132.543%
Depth of quadtree: 7
Number of nodes in quadtree: 341
Compressed image saved to: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\vergil_compressed.jpg
GIF saved to: C:\Users\david\Desktop\Documents\itb\jurusan\sem 4\stima\Tucil2_13523083_13523084\test\vergil_compressed.gif
=====
```

Original	Compressed

#### 5. firefly.jpg (Variance Threshold, Large Block Size)

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly.jpg
-> Error Method Choice: 1
-> Threshold Value: 0.001
-> Target Compression Ratio: 0
-> Minimum Block Size: 40
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed2.jpg
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed2.gif
=====
Are you sure?
Press ENTER to continue or CTRL+C to cancel...
=====
```

```
=====
Image saved successfully.
=====
Image compression completed in 5716 milliseconds.
GIF creation completed in 18444 milliseconds.
Previous image size: 35.0068 KB
Compressed image size: 159.687 KB
Compression ratio: 456.158%
Depth of quadtree: 7
Number of nodes in quadtree: 7965
Compressed image saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed2.jpg
GIF saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\firefly_compressed2.gif
=====
```

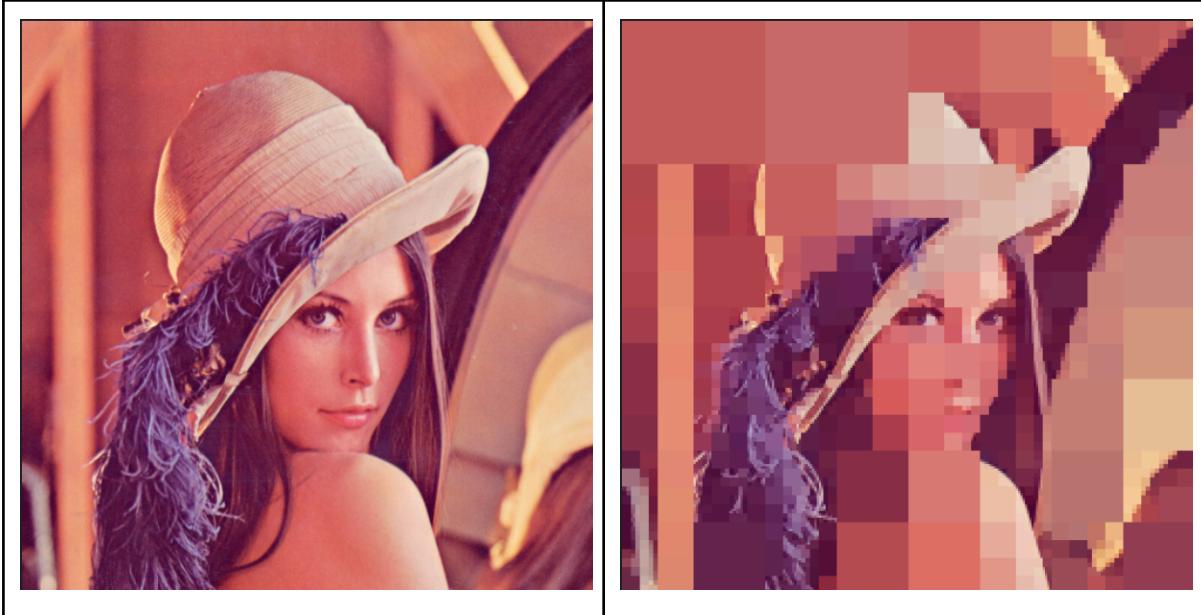


## 6. lena.png (SSIM)

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\lena.png
-> Error Method Choice: 5
-> Threshold Value: -0.1
-> Target Compression Ratio: 0
-> Minimum Block Size: 10
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\lena_compressed2.png
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\lena_compressed2.gif
=====
Are you sure?
```

```
=====
compressing...
=====
Image saved successfully.
=====
Image compression completed in 34724 milliseconds.
GIF creation completed in 58248 milliseconds.
Previous image size: 462.726 KB
Compressed image size: 31.5732 KB
Compression ratio: 6.82332%
Depth of quadtree: 8
Number of nodes in quadtree: 5473
Compressed image saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\lena_compressed2.png
GIF saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\lena_compressed2.gif
=====
```





7. dante.png

```
=====
You have provided the following information:
-> Absolute Path to Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\dante.png
-> Error Method Choice: 4
-> Threshold Value: 1
-> Target Compression Ratio: 0
-> Minimum Block Size: 80
-> Absolute Path to Output Image: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\dante_compressed.png
-> Absolute Path to Output GIF: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\dante_compressed.gif
=====
Are you sure?
```

```
=====
Image saved successfully.
=====
Image compression completed in 62276 milliseconds.
GIF creation completed in 137896 milliseconds.
Previous image size: 615.067 KB
Compressed image size: 82.0996 KB
Compression ratio: 13.3481%
Depth of quadtree: 7
Number of nodes in quadtree: 18801
Compressed image saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\dante_compressed.png
GIF saved to: C:\Users\luthf\Documents\INSTITUT TEKNOLOGI BANDUNG\SEMESTER 4\Tucil2_13523083_13523084\test\dante_compressed.gif
=====
```

Original	Compressed

# BAB IV: ANALISIS

## 1. Kompleksitas Algoritma

Untuk memperdalam analisis terhadap kompleksitas algoritma, pada bab sebelumnya telah dijelaskan algoritma dan alur program yang telah dibangun. Dalam pembangunan QuadTree, best-casenya ketika warna rata-rata gambar sudah langsung memenuhi threshold yang telah dipasang, sehingga dapat disimpulkan bahwa kompresi ini  $\Omega(N)$  dalam sisi kompleksitas waktu, tidak diperlukan pemotongan, namun perlu diuji terlebih dahulu untuk *threshold*-nya. Pada worst case, dengan N sebagai jumlah pixel, dilakukan  $N/4$  terus menerus hingga tiap pixel memiliki node tersendiri. Setiap pixel diproses warna rata-ratanya, sehingga diproses sebanyak N, lalu disusun kembali dari leaf menuju parent node, sehingga dilakukan sebanyak jumlah yang sama dengan metode pemisahan. Dari itu, dapat disimpulkan bahwa pemrosesan worst case dari algoritma QuadTree adalah  $O(N (\log_2 N)^2)$ .

## 2. Rasio Kompresi

Dalam implementasi kompresi gambar dengan metode QuadTree, program telah berhasil mengecilkan ukuran file image dengan threshold yang dipilih. QuadTree menyederhanakan gambar, sehingga kompresi PNG lebih efektif dalam mengurangi ukuran file. Dalam segi kompleksitas, algoritma QuadTree sangat terpengaruh oleh kompleksitas ruang. Gambar dengan noise yang banyak dibandingkan gambar yang relatif datar akan memiliki durasi pemrosesan dan rasio kompresi yang berbeda. Dapat dilihat dari pengujian, bahwa gambar dan tingkat kompresinya tergantung keseragaman pixel. Selain itu, rata-rata dalam pengujian, compression JPG terganggu, menghasilkan file yang jauh lebih besar dari original.

## BAB V: BONUS

### 1. Structural Similarity Index Threshold (SSIM)

Structural Similarity Index (SSIM) adalah sebuah metrik yang digunakan untuk mengukur tingkat kemiripan antara dua gambar. SSIM antara dua gambar dinilai berdasarkan tiga komponen utama: *Luminance*, *Contrast*, *Structure*. Dari ketiga komponen tersebut, dibandingkan seberapa mirip tingkat kecerahan, terang-gelap, dan tekstur loka antara dua gambar. Rumus umum dari SSIM antara matriks kanal satu warna X dan Y adalah sebagai berikut:

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

Pada rumus tersebut,  $C_1$  diberikan oleh  $(K1 \times L)^2$  dan  $C_2$  diberikan oleh  $(K2 \times L)^2$ . Kedua variabel tersebut digunakan untuk menstabilkan perhitungan, khususnya menghindari pembagian dengan nol saat nilai-nilai dalam perhitungan sangat kecil. Menurut makalah orisinal SSIM oleh Wang et al. (2004), konstanta  $K_1$  dan  $K_2$  yang optimal adalah 0.01 dan 0.03, dan  $L$  diberikan sebagai range warna yang mungkin, atau pada program ini, nilai pada matriks. SSIM akan diimplementasikan pada tiap kanal warna dan digabung dengan rumus sebagai berikut:

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Ketika dipasang sebagai *threshold*, program akan menggunakan SSIM untuk membandingkan antara balok berwarna datar berdasarkan warna rata-rata balok tersebut dengan gambar aslinya. Jika balok tersebut masih belum mirip, maka balok tersebut akan dipisahkan lagi. Pada program ini, gambar melakukan pemisahan ketika lebih besar dari *threshold*, tetapi pada perhitungan *threshold* dengan SSIM, program diharapkan melakukan pemisahan ketika lebih kecil dari *threshold*, atau belum semirip yang diinginkan pengguna. Oleh karena itu, pada implementasi, SSIM diberikan nilai negatif agar sesuai dengan program.

### 2. Target Percentage Compression

Threshold percentage ditemukan dengan mengimplementasikan binary search pada program, dengan berulang kali membangun image dengan threshold yang dinamis. Pada awalnya, sesuai dengan metode *threshold* yang dipilih, setiap metode mempunyai nilai minimum yang mungkin dan maksimum yang mungkin. Setelah menentukan kedua tumpuan sebagai min dan max pada binary search, akan diambil nilai tengah dan melakukan penyimpanan kompresi gambar dengan threshold tersebut. Jika terlalu besar, ulangi binary search, dengan nilai tengah tersebut sebagai max. Jika terlalu kecil, ulangi binary search, dengan nilai tengah tersebut sebagai min.

Program akan mengulangi binary search hingga mencapai ujung jumlah iterasi, atau sudah cukup mendekati rasio yang diharapkan pengguna.

### 3. GIF Process

Pembentukan GIF bertujuan untuk memperlihatkan pembagian tiap node menjadi 4. Maka, setiap kedalaman pada *quadtree* yang dibentuk dibuat menjadi satu gambar. Setelah semua gambar terkumpulkan, gambar-gambar (dalam animasi, seringkali disebut frames) ini disusun secara sekuensial dan jeda waktu seragam agar menunjukkan proses kompresi quadtree secara jelas.

Dalam proses penyimpanan GIF ini, hal penting yang harus diperhatikan adalah bagaimana GIF menyimpan warna. Berbeda dengan gambar bertipe png dan jpg/ jpeg yang menyimpan warna setiap pixelnya dengan nilai RGB 24-bit, GIF menyimpan warna dengan menggunakan color palette yang berisikan maksimum 256 warna RGB 24-bit dan mencocokkan warna setiap pixel dengan index warna yang ada pada color palette tersebut. Hal ini secara efektif mengurangi berat gambar karena setiap pixel hanya perlu menyimpan 8-bit untuk merepresentasikan warnanya,  $\frac{1}{3}$  dari jumlah pada file JPG dan PNG.

Proses konversi gambar 24-bit ke 8-bit ini dibantu dengan adanya kelas Bucket, yang berfungsi untuk membantu proses *color quantization*, yaitu membatasi jumlah warna ke jumlah tertentu, seringkali hasilnya disebut sebagai *color palette*. Proses *quantization* ini dilakukan dengan metode *median-cut quantization*, yang membagi *color space* ke jumlah tertentu. Proses pembagian *color space* ini juga menggunakan algoritma divide and conquer, warna-warna padanya diurutkan berdasarkan range terbesar dari suatu kanal, lalu dibagi menjadi dua *color space*. Proses ini diulang terus menerus hingga mencapai sebanyak *color space* yang diinginkan. Terakhir, ambil warna rata-rata dari *color space* tersebut sebagai warna yang disimpan pada color palette. Selelah *color palette* siap, hal terakhir adalah untuk mencocokkan warna asal ke warna yang terdapat pada palette. Hal ini dilakukan dengan menggunakan *euclidean distance* terdekat dari semua warna pada *color palette*.

Selain itu, gambar dengan tipe file GIF dapat di-layer atau ditumpuk untuk menampilkan suatu animasi. Oleh karena itu, gambar-gambar yang didapatkan dari setiap depth perlu dikonversikan ke gambar 8-bit, lalu ditambahkan satu-per-satu pada file GIF dimulai dari depth 0 hingga depth terbesar. Terakhir, pengkodean GIF memiliki fitur extension untuk berbagai kegunaan, salah satunya adalah extension “NETSCAPE2.0” untuk memungkinkan adanya looping pada GIF.

## LAMPIRAN

Pranala ke repository yang berisi kode program.

[https://github.com/koinen/Tucil2\\_13523083\\_13523084](https://github.com/koinen/Tucil2_13523083_13523084)

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2. Program berhasil dijalankan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4. Mengimplementasi seluruh metode perhitungan error wajib	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5. [Bonus] Implementasi persentasi kompresi sebagai parameter tambahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompreksi Gambar	<input checked="" type="checkbox"/>	<input type="checkbox"/>
8. Program dan laporan dibuat (kelompok) sendiri	<input checked="" type="checkbox"/>	<input type="checkbox"/>

# REFERENSI

Median Cut Quantization - [http://web.cs.wpi.edu/~matt/courses/cs563/talks/color\\_quant/CQindex.html](http://web.cs.wpi.edu/~matt/courses/cs563/talks/color_quant/CQindex.html)

giflib documentation - [https://giflib.sourceforge.net/gif\\_lib.html](https://giflib.sourceforge.net/gif_lib.html)

giflib example usage - <https://gist.github.com/suzumura-ss/a5e922994513e44226d33c3a0c2c60d1>

GIF file format - [https://en.wikipedia.org/wiki/GIF#Animated\\_GIF](https://en.wikipedia.org/wiki/GIF#Animated_GIF)

Structural Similarity Index - [https://www.researchgate.net/publication/342435717\\_Understanding\\_SSIM](https://www.researchgate.net/publication/342435717_Understanding_SSIM)