LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Kompresi Gambar Dengan Metode Quadtree

Disusun Oleh:
Adinda Putri
13523071
K-02

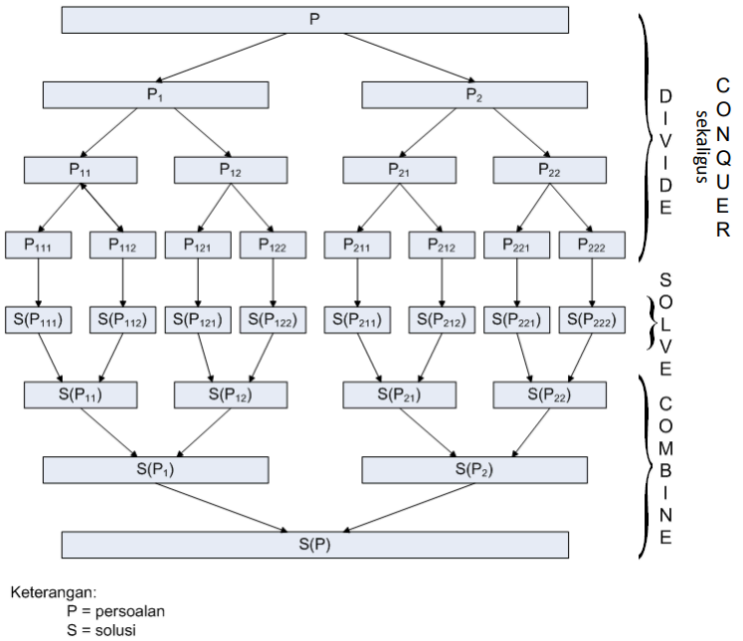
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

1. Pendahuluan

1.1. Algoritma Divide and Conquer

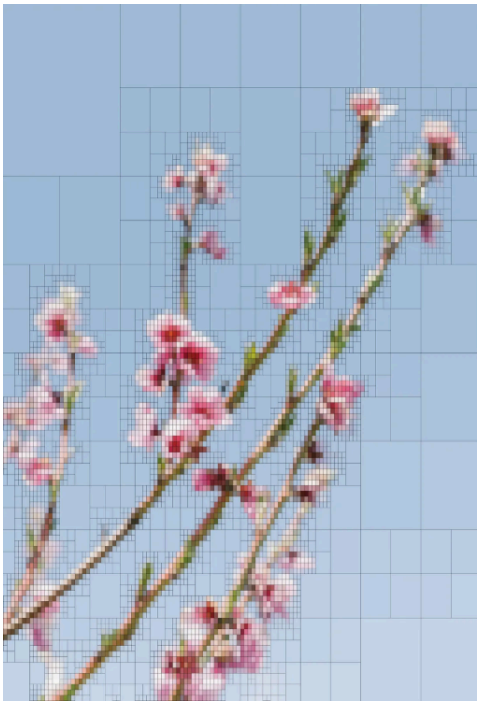
Metode *Divide and Conquer* merupakan salah satu strategi dasar dalam perancangan algoritma pemrograman yang bertujuan untuk menyederhanakan penyelesaian masalah kompleks. Strategi ini membagi masalah utama menjadi beberapa sub-masalah yang lebih kecil dan menyelesaikan masing-masing sub-masalah secara terpisah, kemudian menggabungkan hasilnya untuk membentuk solusi akhir. [1] Tiga tahap utamanya adalah *divide* (pembagian masalah), *conquer* (penyelesaian sub-masalah, sering kali secara rekursif), dan *combine* (penggabungan hasil).

Dalam algoritma pemrograman, *Divide and Conquer* sangat efektif diterapkan pada berbagai permasalahan seperti pengurutan (misalnya *Merge Sort* dan *Quick Sort*), pencarian elemen (seperti *Binary Search*), penghitungan eksponen, hingga pemrosesan matriks dan pembagian polinomial. Karena strukturnya yang mendukung rekursi dan optimisasi, strategi ini menjadi fondasi penting dalam membangun algoritma yang mangkus dan sangkil. Dengan membagi masalah secara sistematis dan memanfaatkan pola penyelesaian berulang, *Divide and Conquer* tidak hanya menyederhanakan proses pemrograman, tetapi juga meningkatkan performa algoritma secara keseluruhan.



Gambar 1. Simulasi Algoritma Divide and Conquer
(Sumber: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

1.2. Metode Quadtree dalam Kompresi Gambar



Gambar 2. Quadtree dalam Kompresi Gambar
(Sumber: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00)

Salah satu struktur data yang dapat diimplementasikan untuk kompresi gambar adalah Quadtree. Quadtree merupakan struktur data pohon yang setiap simpulnya memiliki paling banyak empat anak. Dalam pengolahan gambar, Quadtree membagi blok piksel menjadi empat blok piksel yang lebih kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat blok piksel, lalu memeriksa keseragaman berdasarkan analisis sistem warna RGB. Pemeriksaan keseragaman tersebut dilakukan dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada masing-masing piksel dalam blok tersebut. Jika bagian gambar tersebut tidak seragam, bagian tersebut akan terus dibagi sampai mencapai tingkat keseragaman tertentu atau ukuran blok minimum yang ditentukan.

Secara teknis, Quadtree direpresentasikan dengan simpul dengan maksimal empat anak. Simpul daun merepresentasikan area yang seragam, sedangkan simpul internal merepresentasikan area yang masih membutuhkan pembagian blok. Setiap simpul menyimpan informasi posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Karena pembagian blok berhenti setelah mencapai keseragaman tertentu, Quadtree dapat menghilangkan redundansi dan menghasilkan pengkodean data gambar yang lebih efisien.

2. Algoritma Program

2.1 Algoritma Image Compression

Algoritma Divide and Conquer untuk Image Compression dengan Quadtree secara garis besar terdiri dari tiga langkah utama, yaitu inisialisasi dan persiapan data, perbandingan nilai error metric dengan ambang batas (*threshold*), serta pembagian blok gambar secara rekursif menjadi empat bagian hingga mencapai kondisi berhenti. Berikut proses algoritma Divide and Conquer untuk Image Compression dengan Quadtree:

1 .Menginput gambar yang akan dikompresi.
2 .Mengolah data gambar dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB.
3. Menentukan satu metode perhitungan error dari metode-metode berikut: variance, mean absolute deviation (MAD), max pixel difference, entropy, dan structural similarity index (SSIM).
4. Menentukan variansi threshold dan minimum block size.
5. Menghitung nilai error dan menormalisasi warna blok awal dengan menghitung rata-rata nilai RGB blok.
6. Membandingkan error tersebut dengan variansi threshold. Jika error tersebut lebih besar dari variansi threshold, ukuran blok piksel lebih besar dari minimum block size, dan ukuran blok piksel setelah dibagi menjadi empat tidak kurang dari minimum block size, maka blok piksel tersebut dibagi menjadi empat sub-blok lebih kecil,, lalu pembagian blok piksel dan langkah 5-6 dilakukan secara rekursif untuk masing-masing sub-blok hingga salah satu kondisi di atas tidak terpenuhi.
7. Jika salah satu kondisi di atas tidak terpenuhi, pembagian blok dihentikan.
8. Menyimpan hasil sebagai file terkompresi dan menampilkan persentase kompresi.

2.2 Pseudocode Algoritma

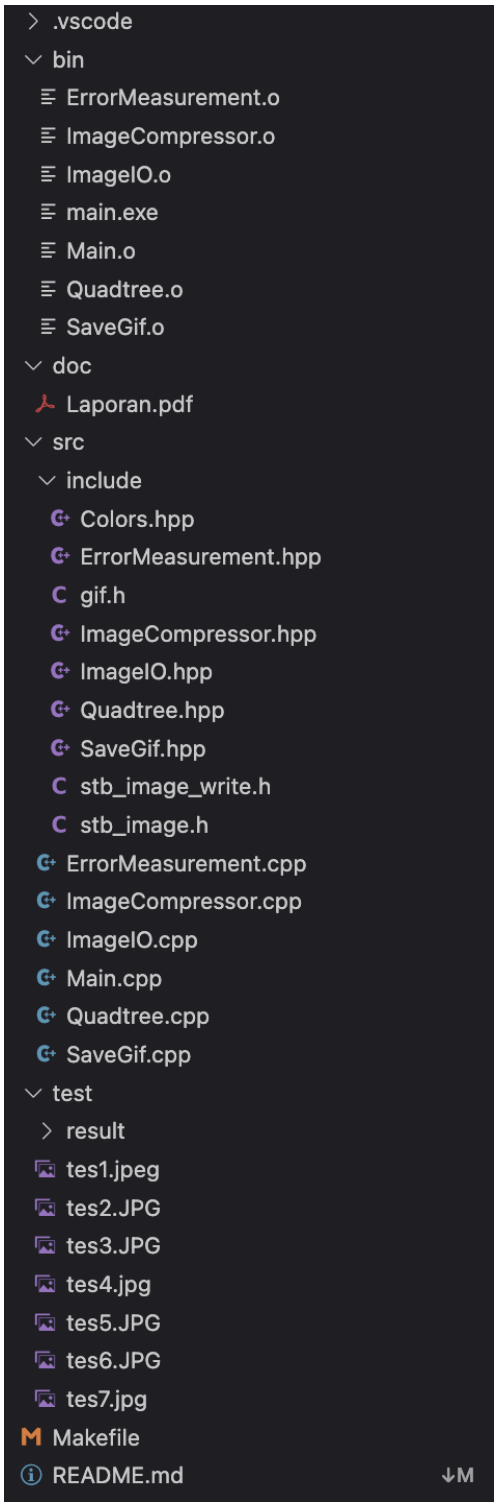
Pseudocode dari Algoritma Divide and Conquer untuk Image Compression dengan Quadtree adalah sebagai berikut.

```
ALGORITMA KompresiGambar
-------------------------------------------------
Input: fileGambar
Output: fileGambarTerkompresi, persentaseKompresi

function KompresiGambar()
    // Menginput gambar
    fileGambar ← bacaGambarDariFile()

    // Mengolah data gambar dalam format matriks piksel
    matriksPiksel ← konversiKeMatriksRGB(fileGambar)
```

```
    // Memilih metode perhitungan error
    metode ← pilihMetodeVariansi()
    // Metode perhitungan error: 'variance', 'MAD', 'max_diff', 'entropy', 'SSIM'

    // Menentukan threshold dan minimum block size
    threshold ← inputThreshold()
    minBlockSize ← inputMinBlockSize()

    // Melakukan image processing blok secara rekursif
    hasil ← prosesBlok(matriksPiksel, threshold, minBlockSize, metode)

    // Menyimpan hasil dan tampilkan persentase kompresi
    simpanGambar(hasil)
    persentase ← hitungPersentaseKompresi(fileGambar, hasil)
    print("Persentase Kompresi: " + persentase + "%")
end function


function prosesBlok(blok, threshold, minBlockSize, metode)
    // Menghitung variansi dan rata-rata warna blok
    variansi ← hitungVariansi(blok, metode)
    rataRataRGB ← hitungRataRataRGB(blok)
    blokNormalisasi ← isiBlokDenganWarna(blok, rataRataRGB)

    // Mengecek apakah blok perlu dibagi
    if variansi > threshold and ukuran(blok) > minBlockSize and ukuran(blok)/2 ≥
minBlockSize then
        // Membagi blok menjadi 4 sub-blok
        subBlokList ← bagiMenjadiEmpat(blok)

        // Rekursif ke tiap sub-blok
        for each subBlok in subBlokList do
            subBlok ← prosesBlok(subBlok, threshold, minBlockSize, metode)
        end for

        // Gabungkan hasil dari sub-blok
        return gabungSubBlok(subBlokList)
    else
        // Tidak dibagi, menggunakan warna rata-rata
        return blokNormalisasi
    end if
end function
```

3. Source Code Program

Pada program ini, image compressing dengan Quadtree dibuat dengan paradigma pemrograman berorientasi objek. Program didekomposisi menjadi beberapa file header dan file implementasi dari class-class sesuai spesialisasinya. Untuk lebih jelasnya, berikut merupakan struktur folder beserta penjelasan masing-masing file header .hpp dan file implementasi .cpp.



Gambar 3. Folder Structure
(Sumber: Penulis)

3.1 Colors.hpp

Berikut ini merupakan source code beserta tabel attribute dan method yang terdapat dalam struct di Colors.hpp.

Colors.hpp

```
#ifndef __COLORS_HPP__
#define __COLORS_HPP__

struct Color {
    int r, g, b;
    Color() : r(0), g(0), b(0) {}
    Color(int red, int green, int blue) : r(red), g(green), b(blue) {}
};

#endif
```

Attribute	Tipe Data	Keterangan
r	public int	Attribut warna merah
g	public int	Atribut warna hijau
b	public int	Atribut warna biru

Method	Returned Data Type	Keterangan
Color	-	Constructor default
Color	-	Constructor dengan parameter

3.2 ErrorMeasurement.hpp dan ErrorMeasurement.cpp

Berikut ini merupakan source code beserta tabel attribute dan method yang terdapat dalam class di ErrorMeasurement.hpp dan ErrorMeasurement.cpp.

ErrorMeasurement.hpp

```
#ifndef __ERRORMEASUREMENT_HPP__
#define __ERRORMEASUREMENT_HPP__

#include <vector>
```

```
#include "Colors.hpp"

class ErrorMeasurement {
public:
    static const int MAX_COLOR = 256;

    static double variance(const std::vector<std::vector<Color>>& pixels, int x, int y,
int width, int height);
    static double mad(const std::vector<std::vector<Color>>& pixels, int x, int y, int
width, int height);
    static double maxPixelDifference(const std::vector<std::vector<Color>>& pixels, int
x, int y, int width, int height);
    static double entropy(const std::vector<std::vector<Color>>& pixels, int x, int y,
int width, int height);
    static double ssim(const std::vector<std::vector<Color>>& pixels, int x, int y, int
width, int height);
};

#endif
```

Attribute	Tipe Data	Keterangan
MAX_COLOR = 256	public static const int	Konstanta maksimum nilai kanal R, G, dan B

Method	Returned Data Type	Keterangan
variance	public static double	Untuk menghitung error dengan metode variansi
mad	public static double	Untuk menghitung error dengan metode MAD
maxPixelDifference	public static double	Untuk menghitung error dengan metode max pixed difference
entropy	public static double	Untuk menghitung error dengan metode entropy
ssim	public static double	Untuk menghitung error dengan metode SSIM

ErrorMeasurement.cpp

```cpp
#include "ErrorMeasurement.hpp"
#include <cmath>
#include <vector>

double ErrorMeasurement::variance(const std::vector<std::vector<Color>>& pixels, int x,
int y, int width, int height) {
    int count = width * height;
    double sumR = 0, sumG = 0, sumB = 0;

    for (int i = y; i < y + height; i++)
        for (int j = x; j < x + width; j++) {
            sumR += pixels[i][j].r;
            sumG += pixels[i][j].g;
            sumB += pixels[i][j].b;
        }

    double avgR = sumR / count;
    double avgG = sumG / count;
    double avgB = sumB / count;

    double var = 0.0;
    for (int i = y; i < y + height; i++)
        for (int j = x; j < x + width; j++) {
            var += std::pow(pixels[i][j].r - avgR, 2)
                 + std::pow(pixels[i][j].g - avgG, 2)
                 + std::pow(pixels[i][j].b - avgB, 2);
        }

    return var / count;
}

double ErrorMeasurement::mad(const std::vector<std::vector<Color>>& pixels, int x, int
y, int width, int height) {
    int count = width * height;
    double sumR = 0, sumG = 0, sumB = 0;

    for (int i = y; i < y + height; i++)
```

```
        for (int j = x; j < x + width; j++) {
            sumR += pixels[i][j].r;
            sumG += pixels[i][j].g;
            sumB += pixels[i][j].b;
        }

    double avgR = sumR / count;
    double avgG = sumG / count;
    double avgB = sumB / count;

    double mad = 0.0;
    for (int i = y; i < y + height; i++)
        for (int j = x; j < x + width; j++) {
            mad += std::abs(pixels[i][j].r - avgR)
                 + std::abs(pixels[i][j].g - avgG)
                 + std::abs(pixels[i][j].b - avgB);
        }

    return mad / count;
}

double ErrorMeasurement::maxPixelDifference(const std::vector<std::vector<Color>>&
pixels, int x, int y, int width, int height) {
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;
    for (int i = y; i < y + height; i++)
        for (int j = x; j < x + width; j++) {
            const Color& c = pixels[i][j];
            if (c.r < minR) minR = c.r;
            if (c.g < minG) minG = c.g;
            if (c.b < minB) minB = c.b;
            if (c.r > maxR) maxR = c.r;
            if (c.g > maxG) maxG = c.g;
            if (c.b > maxB) maxB = c.b;
        }
    double diffR = maxR - minR;
    double diffG = maxG - minG;
    double diffB = maxB - minB;
    return (diffR + diffG + diffB) / 3.0;
}

double ErrorMeasurement::entropy(const std::vector<std::vector<Color>>& pixels, int x,
```

```
int y, int width, int height) {
    const int count = width * height;
    std::vector<int> histR(MAX_COLOR, 0), histG(MAX_COLOR, 0), histB(MAX_COLOR, 0);

    for (int i = y; i < y + height; ++i)
        for (int j = x; j < x + width; ++j) {
            const Color& c = pixels[i][j];
            ++histR[c.r];
            ++histG[c.g];
            ++histB[c.b];
        }

    auto computeEntropyChannel = [count](const std::vector<int>& hist) -> double {
        double ent = 0.0;
        for (int i = 0; i < MAX_COLOR; ++i) {
            if (hist[i] > 0) {
                double p = static_cast<double>(hist[i]) / count;
                ent -= p * std::log2(p);
            }
        }
        return ent;
    };

    return (computeEntropyChannel(histR) + computeEntropyChannel(histG) +
computeEntropyChannel(histB)) / 3.0;
}

double ErrorMeasurement::ssim(const std::vector<std::vector<Color>>& pixels, int x, int
y, int width, int height) {
    const double L = 255.0;
    const double C1 = (0.01 * L) * (0.01 * L);
    const double C2 = (0.03 * L) * (0.03 * L);
    int N = width * height;

    if (N == 0) return 1.0;

    std::vector<double> weights = {0.2125, 0.7154, 0.0721};
    double totalWeight = weights[0] + weights[1] + weights[2];

    double totalSSIM = 0.0;

    for (int channel = 0; channel < 3; ++channel) {
```

```
        double mu = 0.0;
        double sigma = 0.0;

        for (int i = y; i < y + height; ++i) {
            for (int j = x; j < x + width; ++j) {
                double val = 0.0;
                if (channel == 0) val = pixels[i][j].r;
                else if (channel == 1) val = pixels[i][j].g;
                else if (channel == 2) val = pixels[i][j].b;
                mu += val;
            }
        }
        mu /= N;

        for (int i = y; i < y + height; ++i) {
            for (int j = x; j < x + width; ++j) {
                double val = 0.0;
                if (channel == 0) val = pixels[i][j].r;
                else if (channel == 1) val = pixels[i][j].g;
                else if (channel == 2) val = pixels[i][j].b;
                sigma += (val - mu) * (val - mu);
            }
        }
        sigma /= N;

        double mu_y = mu;
        double sigma_y = 0.0;
        double covariance = 0.0;

        double numerator = (2 * mu * mu_y + C1) * (2 * covariance + C2);
        double denominator = (mu * mu + mu_y * mu_y + C1) * (sigma + sigma_y + C2);
        double ssim = (denominator == 0.0) ? 1.0 : numerator / denominator;

        totalSSIM += ssim * weights[channel];
    }

    double ssim_avg = totalSSIM / totalWeight;
    return 1.0 - ssim_avg;
}
```

3.3 Quadtree.hpp dan Quadtree.cpp

Berikut ini merupakan source code beserta tabel attribute dan method yang terdapat dalam class di Quadtree.hpp dan Quadtree.cpp

Quadtree.hpp

```
#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include <vector>
#include "Colors.hpp"

class QuadtreeNode {
public:
    int x, y;
    int width, height;
    int depth;
    Color color;
    bool is_leaf;
    double error;

    QuadtreeNode* children[4];

    QuadtreeNode(int x, int y, int width, int height);
    ~QuadtreeNode();
};

class Quadtree {
private:
    QuadtreeNode* root;
    int width;
    int height;

    int countNodes(const QuadtreeNode* node) const;
    int maxDepth(const QuadtreeNode* node) const;

public:
    Quadtree(QuadtreeNode* root, int width, int height);
    ~Quadtree();

    QuadtreeNode* getRoot() const;
    int getWidth() const;
    int getHeight() const;
```

```
    int countNodes() const;
    int maxDepth() const;

    std::vector<std::vector<Color>> renderToPixels() const;
    std::vector<std::vector<Color>> renderAtDepth(int depthLevel) const;
};

#endif
```

class QuadtreeNode

Attribute	Tipe Data	Keterangan
x	public int	Indeks colom blok piksel
y	public int	Indeks row blok piksel
width	public int	Lebar blok piksel
height	public int	Tinggi blok piksel
depth	public int	Kedalaman node
color	public Color	Rata-rata nilai RGB
is_leaf	public bool	Predikat boolean untuk daun
error	public double	Untuk nilai error
children[4]	public QuadtreeNode*	Array of pointer ke keempat sub-blok

Method	Returned Data Type	Keterangan
QuadtreeNode	-	Constructor dengan parameter
~QuadtreeNode	-	Destruktor

class Quadtree

Attribute	Tipe Data	Keterangan
root	private QuadtreeNode*	Pointer ke node akar

width	private int	Lebar gambar
height	private int	Tinggi gambar

Method	Returned Data Type	Keterangan
Quadtree	-	Constructor dengan parameter
~Quadtree	-	Destructor
getRoot const	public QuadTreeNode*	Mengakses pointer ke akar pohon
getWidth const	public int	Mengambil lebar pohon
getHeight const	public int	Mengambil tinggi pohon
countNode const	public int	Menghitung jumlah node dalam pohon
maxDepth const	public int	Menghitung kedalaman maksimal pohon
renderToPixels const	public std::vector<std::vector<Color >>	Render seluruh pohon menjadi vektor warna 2D
renderAtDepth const	public std::vector<std::vector<Color >>	Render poho menjadi vektor warna 2D hingga kedalaman tertentu
countNodes const	private int	Menghitung jumlah node dalam pohon
maxDepth const	private int	Menghitung kedalaman maksimal pohon

Quadtree.cpp

```
#include "Quadtree.hpp"
#include <algorithm>

QuadtreeNode::QuadtreeNode(int x, int y, int width, int height)
    : x(x), y(y), width(width), height(height), depth(0),
      color(0, 0, 0), is_leaf(false), error(0.0) {
```

```
    for (int i = 0; i < 4; ++i)
        children[i] = nullptr;
}

QuadtreeNode::~QuadtreeNode() {
    for (int i = 0; i < 4; ++i) {
        delete children[i];
        children[i] = nullptr;
    }
}

Quadtree::Quadtree(QuadtreeNode* root, int width, int height)
    : root(root), width(width), height(height) {}

Quadtree::~Quadtree() {
    delete root;
}

static void fillBlock(std::vector<std::vector<Color>>& pixels, QuadtreeNode* node) {
    if (!node) return;

    if (node->is_leaf) {
        for (int i = node->y; i < node->y + node->height; ++i)
            for (int j = node->x; j < node->x + node->width; ++j)
                pixels[i][j] = node->color;
    } else {
        for (int i = 0; i < 4; ++i)
            fillBlock(pixels, node->children[i]);
    }
}

std::vector<std::vector<Color>> Quadtree::renderToPixels() const {
    std::vector<std::vector<Color>> result(height, std::vector<Color>(width));
    fillBlock(result, root);
    return result;
}

static void drawWithDepth(QuadtreeNode* node, int targetDepth,
std::vector<std::vector<Color>>& canvas) {
    if (!node) return;

    if (node->depth == targetDepth || node->is_leaf) {
```

```
        for (int i = node->y; i < node->y + node->height; ++i)
            for (int j = node->x; j < node->x + node->width; ++j)
                canvas[i][j] = node->color;
    } else {
        for (int i = 0; i < 4; ++i)
            drawWithDepth(node->children[i], targetDepth, canvas);
    }
}

std::vector<std::vector<Color>> Quadtree::renderAtDepth(int depthLevel) const {
    std::vector<std::vector<Color>> result(height, std::vector<Color>(width, Color(200,
200, 200)));;
    drawWithDepth(root, depthLevel, result);
    return result;
}

int Quadtree::countNodes(const QuadtreeNode* node) const {
    if (!node) return 0;
    if (node->is_leaf) return 1;

    int total = 1;
    for (int i = 0; i < 4; ++i)
        total += countNodes(node->children[i]);
    return total;
}

int Quadtree::maxDepth(const QuadtreeNode* node) const {
    if (!node || node->is_leaf) return 1;

    int depth = 0;
    for (int i = 0; i < 4; ++i)
        depth = std::max(depth, maxDepth(node->children[i]));
    return depth + 1;
}

int Quadtree::countNodes() const {
    return countNodes(root);
}

int Quadtree::maxDepth() const {
    return maxDepth(root);
}
```

```
QuadtreeNode* Quadtree::getRoot() const {
    return root;
}

int Quadtree::getWidth() const {
    return width;
}

int Quadtree::getHeight() const {
    return height;
}
```

3.4 ImageIO.hpp dan ImageIO.cpp

Berikut ini merupakan source code beserta tabel attribute dan method yang terdapat dalam class di ImageIO.hpp dan ImageIO.cpp.

ImageIO.hpp

```
#ifndef IMAGE_IO_HPP
#define IMAGE_IO_HPP

#include <string>
#include <vector>
#include "Colors.hpp"
#include "Quadtree.hpp"

class ImageIO {
public:
    static bool loadImage(const std::string &path, std::vector<std::vector<Color>>
&pixelData);
    static bool saveImage(const std::string &path, Quadtree *tree, bool drawOutline =
false);
    static long getFileSize(const std::string &path);
};

#endif
```

Method	Returned Data Type	Keterangan

loadImage	public static boolean	Untuk mengubah gambar menjadi vektor warna 2D
saveImage	public static boolean	Untuk mengubah vektor warna 2D menjadi gambar
getFileSize	public static long	Untuk mendapatkan ukuran gambar

ImageIO.cpp

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

#include "ImageIO.hpp"
#include <iostream>
#include <fstream>
#include <filesystem>

using namespace std;

bool ImageIO::loadImage(const std::string &path, std::vector<std::vector<Color>>
&pixelData) {
    int width, height, channels;
    unsigned char *data = stbi_load(path.c_str(), &width, &height, &channels, 3); //
Paksa jadi RGB (tanpa alpha)
    if (!data) {
        cerr << "Failed to load image: " << path << endl;
        return false;
    }

    pixelData.resize(height, std::vector<Color>(width));
    for (int y = 0; y < height; ++y)
        for (int x = 0; x < width; ++x) {
            int idx = (y * width + x) * 3;
            pixelData[y][x] = Color(data[idx], data[idx + 1], data[idx + 2]);
        }

    stbi_image_free(data);
    return true;
```

```
}

bool ImageIO::saveImage(const std::string &path, Quadtree *tree, bool drawOutline) {
    if (!tree || !tree->getRoot()) {
        cerr << "Invalid quadtree.\n";
        return false;
    }

    int width = tree->getWidth();
    int height = tree->getHeight();
    int channels = 3;
    std::vector<unsigned char> imageData(width * height * channels, 255);

    auto setPixel = [&](int x, int y, const Color &color) {
        int idx = (y * width + x) * 3;
        imageData[idx + 0] = color.r;
        imageData[idx + 1] = color.g;
        imageData[idx + 2] = color.b;
    };

    std::function<void(QuadtreeNode*)> drawNode = [&](QuadtreeNode *node) {
        if (!node) return;
        if (node->is_leaf) {
            for (int y = node->y; y < node->y + node->height; ++y)
                for (int x = node->x; x < node->x + node->width; ++x)
                    setPixel(x, y, node->color);

            if (drawOutline) {
                Color black(0, 0, 0);
                for (int x = node->x; x < node->x + node->width; ++x) {
                    setPixel(x, node->y, black);
                    setPixel(x, node->y + node->height - 1, black);
                }
                for (int y = node->y; y < node->y + node->height; ++y) {
                    setPixel(node->x, y, black);
                    setPixel(node->x + node->width - 1, y, black);
                }
            }
        } else {
            for (int i = 0; i < 4; ++i)
                drawNode(node->children[i]);
        }
```

```
    };

    drawNode(tree->getRoot());

    int success = stbi_write_png(path.c_str(), width, height, channels,
imageData.data(), width * channels);
    if (!success) {
        cerr << "Failed to write image: " << path << endl;
        return false;
    }

    return true;
}

long ImageIO::getFileSize(const std::string &path) {
    return std::filesystem::file_size(path);
}
```

3.5 ImageCompressor.hpp dan ImageCompressor.cpp

Berikut ini merupakan attribute dan method yang terdapat dalam class ImageCompressor.hpp dan ImageCompressor.cpp.

ImageCompressor.hpp

```
#ifndef IMAGE_COMPRESSOR_HPP
#define IMAGE_COMPRESSOR_HPP

#include <string>
#include <vector>
#include <functional>
#include "Colors.hpp"
#include "Quadtree.hpp"
#include "ErrorMeasurement.hpp"

class ImageCompressor {
public:

    ImageCompressor();
    void run();
    ~ImageCompressor() noexcept = default;
```

```
private:
    double threshold;
    int min_block_size;
    std::function<double(const std::vector<std::vector<Color>>&, int, int, int, int)>
errorFunc;

    Color getAverageColor(const std::vector<std::vector<Color>>& image_data, int x, int
y, int width, int height);
    bool shouldDivide(double error, int width, int height);
    QuadtreeNode* build(const std::vector<std::vector<Color>>& image_data, int x, int
y, int width, int heigh, int depth);
    Quadtree* compress(const std::vector<std::vector<Color>>& image_data,
        double target_compression,
        const std::string& tempPath,
        long originalSize,
        std::vector<std::vector<std::vector<Color>>>* gifFrames);
    void setErrorFunction(std::function<double(const std::vector<std::vector<Color>>&,
int, int, int, int)> func);
};

#endif
```

Attribute	Tipe Data	Keterangan
threshold	private double	Menyimpan variansi threshold
min_block_size	private int	Ukuran minimum block piksel

Method	Returned Data type	Keterangan
ImageCompressor	-	Constructor default
~ImageCompressor	-	Destruktor
run	public void	Menangani seluruh alur program yang berkaitan dengan user
errorFunc	private std::function<double(const std::vector<std::vector<Color	Sebagai fungsi error yang digunakan

	>>&, int, int, int, int)>	
getAverageColor	private Color	Menghitung rata-rata warna (RGB) pada suatu blok gambar
shouldDivide	private bool	Menentukan apakah sebuah blok perlu dibagi lagi berdasarkan nilai error
build	private QuadtreeNode*	Membangun struktur pohon berdasarkan hasil kompresi
compress	private Quadtree*	Untuk memproses gambar dan menghasilkan Quadtree hasil kompresi
setErrorFunction	private void	Mengatur fungsi error yang digunakan

ImageCompressor.cpp

```cpp
#include "ImageCompressor.hpp"
#include "ImageIO.hpp"
#include "SaveGif.hpp"
#include <iostream>
#include <cmath>
#include <chrono>
#include <functional>
#include <stdexcept>

ImageCompressor::ImageCompressor()
    : threshold(0), min_block_size(1), errorFunc(ErrorMeasurement::variance) {}

Color ImageCompressor::getAverageColor(const std::vector<std::vector<Color>>&
image_data, int x, int y, int width, int height) {
    long sum_r = 0, sum_g = 0, sum_b = 0;
    for (int i = y; i < y + height; ++i)
        for (int j = x; j < x + width; ++j) {
            sum_r += image_data[i][j].r;
            sum_g += image_data[i][j].g;
            sum_b += image_data[i][j].b;
        }
    int total = width * height;
```

```
    return Color(sum_r / total, sum_g / total, sum_b / total);
}

bool ImageCompressor::shouldDivide(double error, int width, int height) {
    if (width <= min_block_size || height <= min_block_size) return false;
    if (error <= threshold) return false;
    return true;
}

QuadtreeNode* ImageCompressor::build(const std::vector<std::vector<Color>>& image_data,
                                     int x, int y, int width, int height, int depth) {
    QuadtreeNode* node = new QuadtreeNode(x, y, width, height);
    node->depth = depth;

    double error = errorFunc(image_data, x, y, width, height);

    if (!shouldDivide(error, width, height)) {
        node->color = getAverageColor(image_data, x, y, width, height);
        node->is_leaf = true;
        return node;
    }

    node->is_leaf = false;

    int half_width = width / 2;
    int half_height = height / 2;

    node->children[0] = build(image_data, x, y, half_width, half_height, depth + 1);
    node->children[1] = build(image_data, x + half_width, y, width - half_width,
half_height, depth + 1);
    node->children[2] = build(image_data, x, y + half_height, half_width, height -
half_height, depth + 1);
    node->children[3] = build(image_data, x + half_width, y + half_height, width -
half_width, height - half_height, depth + 1);

    return node;
}

Quadtree* ImageCompressor::compress(const std::vector<std::vector<Color>>& image_data,
                                    double target_compression,
                                    const std::string& tempPath,
                                    long originalSize,
```

```
                                    std::vector<std::vector<std::vector<Color>>>*
gifFrames) {
    if (target_compression <= 0.0) {
        // Kompresi biasa tanpa target
        int height = image_data.size();
        int width = image_data[0].size();
        QuadtreeNode* root = build(image_data, 0, 0, width, height, 0);
        return new Quadtree(root, width, height);
    }

    double currentThreshold = this->threshold;
    const double maxThreshold = 1000.0;
    const double step = 5.0;
    const int maxSteps = 200;
    double bestRatio = 0.0;
    Quadtree* bestTree = nullptr;

    for (int stepCount = 0; currentThreshold <= maxThreshold && stepCount < maxSteps;
++stepCount, currentThreshold += step) {
        ImageCompressor tempCompressor;
        tempCompressor.threshold = currentThreshold;
        tempCompressor.min_block_size = min_block_size;
        tempCompressor.setErrorFunction(this->errorFunc);

        Quadtree* tempTree = tempCompressor.compress(image_data, 0.0, "", 0, nullptr);
        ImageIO::saveImage(tempPath, tempTree, false);

        long tempSize = ImageIO::getFileSize(tempPath);
        double ratio = 1.0 - static_cast<double>(tempSize) / originalSize;
        std::cout << "\033[1;36m[OUTPUT]\033[0m [THRESHOLD = " << currentThreshold <<
"] Compression: " << ratio * 100 << "%\n";

        if (ratio >= target_compression) {
            if (bestTree) delete bestTree;
            return tempTree;
        }
        if (ratio > bestRatio) {
            if (bestTree) delete bestTree;
            bestTree = tempTree;
            bestRatio = ratio;
        } else {
            delete tempTree;
```

```
        }

    }

    std::cout << "\033[1;36m[OUTPUT]\033[0m [INFO] Target not reached. Using best
compression: " << bestRatio * 100 << "%\n";
    return bestTree;
}

void ImageCompressor::setErrorFunction(std::function<double(const
std::vector<std::vector<Color>>&, int, int, int, int)> func) {
    errorFunc = func;
}

void ImageCompressor::run() {
    std::string inputPath, outputPath, gifPath;
    int methodChoice = 0;
    double targetCompression = 0.0;
    int saveGifAnswer = -1;
    bool drawOutline = false;

    std::cout << "\033[1;36m[INPUT]\033[0m Enter image path: ";
    std::getline(std::cin, inputPath);
    while (inputPath.empty()) {
        std::cerr << "\033[1;31m[ERROR]\033[0m Image path cannot be empty. Please enter
again: ";
        std::getline(std::cin, inputPath);
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Choose error metric (1-5):\n"
              << "   1. Variance\n   2. MAD\n   3. Max Pixel Difference\n   4.
Entropy\n   5. SSIM\n"
              << "\033[1;36m[INPUT]\033[0m Your choice: ";
    while (!(std::cin >> methodChoice) || methodChoice < 1 || methodChoice > 5) {
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Invalid metric. Please enter 1–5: ";
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Enter threshold (>= 0): ";
    while (!(std::cin >> threshold) || threshold < 0) {
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Threshold must be >= 0. Please re-enter:
```

```
";
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Enter minimum block size (>= 1): ";
    while (!(std::cin >> min_block_size) || min_block_size < 1) {
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Minimum block size must be >= 1. Please
re-enter: ";
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Enter target compression (0-1, 0 to
disable): ";
    while (!(std::cin >> targetCompression) || targetCompression < 0.0 ||
targetCompression > 1.0) {
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Compression must be between 0 and 1.
Please re-enter: ";
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Draw outline? (1 = yes, 0 = no): ";
    int outlineInput;
    while (!(std::cin >> outlineInput) || (outlineInput != 0 && outlineInput != 1)) {
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Please enter 0 or 1: ";
    }
    drawOutline = (outlineInput == 1);
    std::cin.ignore();

    std::cout << "\033[1;36m[INPUT]\033[0m Enter output image path: ";
    std::getline(std::cin, outputPath);
    while (outputPath.empty()) {
        std::cerr << "\033[1;31m[ERROR]\033[0m Output path cannot be empty. Please
enter again: ";
        std::getline(std::cin, outputPath);
    }

    std::cout << "\033[1;36m[INPUT]\033[0m Save as GIF? (1 = yes, 0 = no): ";
    while (!(std::cin >> saveGifAnswer) || (saveGifAnswer != 0 && saveGifAnswer != 1))
{
        std::cin.clear(); std::cin.ignore(10000, '\n');
        std::cerr << "\033[1;31m[ERROR]\033[0m Please enter 0 or 1: ";
    }
```

```
    std::cin.ignore();
    if (saveGifAnswer == 1) {
        std::cout << "\033[1;36m[INPUT]\033[0m Enter GIF path: ";
        std::getline(std::cin, gifPath);
        while (gifPath.empty()) {
            std::cerr << "\033[1;31m[ERROR]\033[0m GIF path cannot be empty. Please
enter again: ";
            std::getline(std::cin, gifPath);
        }
    }

    std::vector<std::vector<Color>> pixelData;
    if (!ImageIO::loadImage(inputPath, pixelData)) {
        std::cerr << "\033[1;31m[ERROR]\033[0m Failed to load input image.\n";
        return;
    }

    switch (methodChoice) {
        case 1: setErrorFunction(ErrorMeasurement::variance); break;
        case 2: setErrorFunction(ErrorMeasurement::mad); break;
        case 3: setErrorFunction(ErrorMeasurement::maxPixelDifference); break;
        case 4: setErrorFunction(ErrorMeasurement::entropy); break;
        case 5: setErrorFunction(ErrorMeasurement::ssim); break;
    }

    auto start_time = std::chrono::high_resolution_clock::now();
    const std::string tempPath = "temp_result.jpg";
    std::vector<std::vector<std::vector<Color>>> gifFrames;
    Quadtree* tree = compress(pixelData, targetCompression, tempPath,
ImageIO::getFileSize(inputPath),
                              gifPath.empty() ? nullptr : &gifFrames);

    ImageIO::saveImage(outputPath, tree, drawOutline);

    if (!gifPath.empty()) {
        for (int d = 1; d <= tree->maxDepth(); ++d) {
            gifFrames.push_back(tree->renderAtDepth(d));
        }

        if (SaveGif::saveGIF(gifPath, gifFrames, 100)) {
            std::cout << "\033[1;36m[OUTPUT]\033[0m GIF saved at: " << gifPath << "\n";
        } else {
```

```
            std::cerr << "\033[1;31m[ERROR]\033[0m Failed to save GIF.\n";
        }
    }

    auto end_time = std::chrono::high_resolution_clock::now();
    double execTime = std::chrono::duration_cast<std::chrono::milliseconds>(end_time -
start_time).count();

    std::cout << "\n\033[1;36m[OUTPUT]\033[0m ========= COMPRESSION REPORT
=========\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Execution time        : " << execTime << "
ms\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Original image size   : " <<
ImageIO::getFileSize(inputPath) / 1024.0 << " KB\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Compressed image size : " <<
ImageIO::getFileSize(outputPath) / 1024.0 << " KB\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Compression percentage: "
              << (1.0 - ImageIO::getFileSize(outputPath) /
static_cast<double>(ImageIO::getFileSize(inputPath))) * 100.0 << "%\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Tree depth            : " <<
tree->maxDepth() << "\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Total nodes           : " <<
tree->countNodes() << "\n";
    std::cout << "\033[1;36m[OUTPUT]\033[0m Output image path     : " << outputPath <<
"\n";

    delete tree;
    std::cin.ignore();
}
```

3.6 SaveGif.hpp dan SaveGIF.cpp

Berikut ini merupakan attribute dan method yang terdapat dalam class SaveGif.hpp dan SaveGif.cpp.

SaveGif.hpp

```
#include <string>
#include <vector>
#include "Colors.hpp"
```

```
class SaveGif {
    public:
        static bool saveGIF(const std::string &gifPath, const
std::vector<std::vector<std::vector<Color>>> &frames, int delayMs);
};
```

Method	Returned Data Type	Keterangan
saveGIF	static boolean	Untuk menyimpan urutan frame gambar hasil dalam format .gif

SaveGif.cpp

```
#include "SaveGif.hpp"
#include <iostream>
#include "gif.h"

bool SaveGif::saveGIF(const std::string &gifPath, const
std::vector<std::vector<std::vector<Color>>> &frames, int delayMs) {
    if (frames.empty()) {
        std::cerr << "[ERROR] No frames provided to save GIF." << std::endl;
        return false;
    }

    int height = frames[0].size();
    int width = frames[0][0].size();

    GifWriter writer;
    if (!GifBegin(&writer, gifPath.c_str(), width, height, delayMs)) {
        std::cerr << "[ERROR] Failed to create GIF at path: " << gifPath << std::endl;
        return false;
    }

    for (const auto& frame : frames) {
        std::vector<uint8_t> frameData(width * height * 4);

        for (int y = 0; y < height; ++y) {
            for (int x = 0; x < width; ++x) {
                int idx = (y * width + x) * 4;
```

```
                frameData[idx + 0] = frame[y][x].r;
                frameData[idx + 1] = frame[y][x].g;
                frameData[idx + 2] = frame[y][x].b;
                frameData[idx + 3] = 255;
            }
        }

        GifWriteFrame(&writer, frameData.data(), width, height, delayMs);
    }

    GifEnd(&writer);
    std::cout << "[INFO] GIF saved successfully to: " << gifPath << std::endl;
    return true;
}
```

3.7 Main.cpp

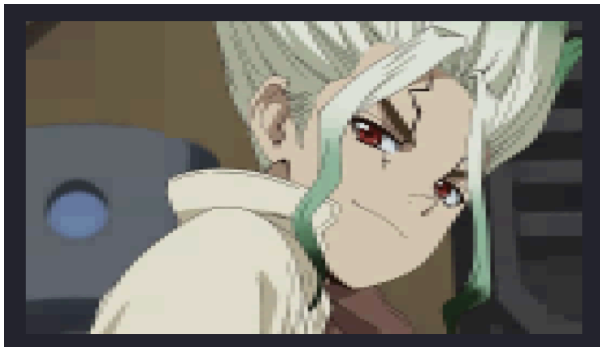
```
#include <iostream>
#include "ImageCompressor.hpp"
int main() {
   std::cout << "============================================\n";
   std::cout << "           Welcome to QuaQua Image          \n";
   std::cout << "       Quadtree -- Divide and Conquer       \n";
   std::cout << "============================================\n\n";
   try {
       ImageCompressor compressor;
       compressor.run();
   } catch (const std::exception& e) {
       std::cerr << "[FATAL ERROR] " << e.what() << "\n";
       return EXIT_FAILURE;
   }
   return EXIT_SUCCESS;
}
```

3. Test Case

Input	Output

```
========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/tes1.jpeg
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 1
[INPUT] Enter threshold (>= 0): 100
[INPUT] Enter minimum block size (>= 1): 4
[INPUT] Enter target compression (0-1, 0 to disable): 0
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes1.jpeg
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes1.gif
[INFO] GIF saved successfully to: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes1.gif
```

```
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes1.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 439 ms
[OUTPUT] Original image size   : 34.5312 KB
[OUTPUT] Compressed image size : 39.1992 KB
[OUTPUT] Compression percentage: -13.5181%
[OUTPUT] Tree depth            : 8
[OUTPUT] Total nodes           : 9297
[OUTPUT] Output image path     : /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes1.jpeg
```

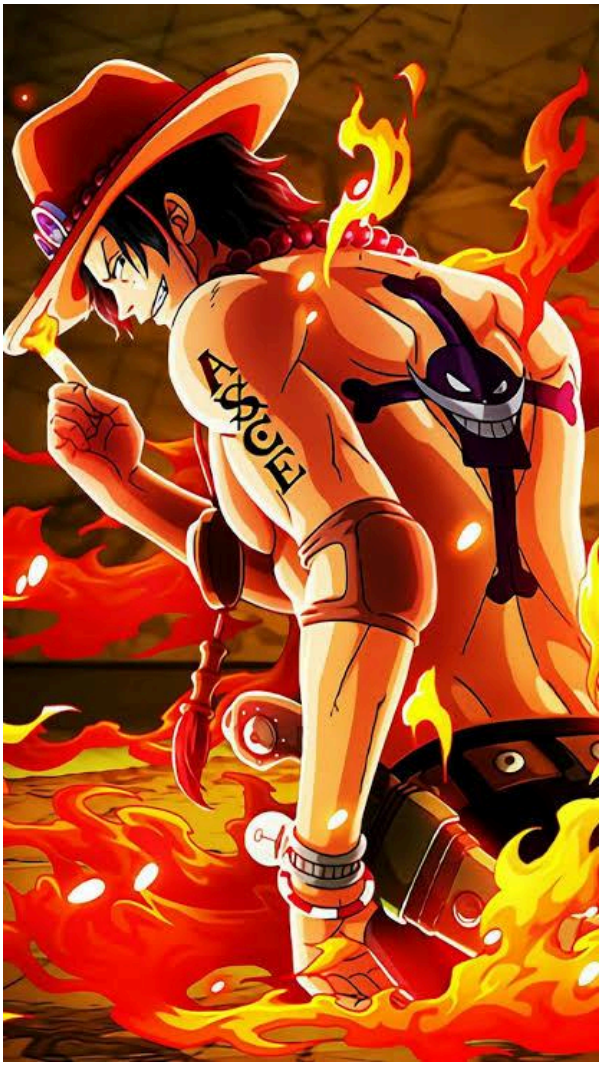


```
[INPUT] Enter image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/tes2.JPG
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 1
[INPUT] Enter threshold (>= 0): 500
[INPUT] Enter minimum block size (>= 1): 4
[INPUT] Enter target compression (0-1, 0 to disable): 0
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes2.JPG
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes2.gif
[INFO] GIF saved successfully to: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes2.gif
```

```
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes2.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 362 ms
[OUTPUT] Original image size   : 35.792 KB
[OUTPUT] Compressed image size : 37.0264 KB
[OUTPUT] Compression percentage: -3.44875%
[OUTPUT] Tree depth            : 8
[OUTPUT] Total nodes           : 8237
[OUTPUT] Output image path     : /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes2.JPG
```

```
==========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
==========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/tes3.JPG
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 2
[INPUT] Enter threshold (>= 0): 0
[INPUT] Enter minimum block size (>= 1): 1
[INPUT] Enter target compression (0-1, 0 to disable): 0
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes3.JPG
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes3.gif
[INFO] GIF saved successfully to: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes3.gif
```

```
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes3.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 655 ms
[OUTPUT] Original image size   : 96.917 KB
[OUTPUT] Compressed image size : 511.979 KB
[OUTPUT] Compression percentage: -428.266%
[OUTPUT] Tree depth            : 10
[OUTPUT] Total nodes           : 243537
[OUTPUT] Output image path     : /Users/adndax/Documents/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/result/tes3.JPG
```



```
==========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
==========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academi
c/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/t
es4.jpg
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 3
[INPUT] Enter threshold (>= 0): 200
[INPUT] Enter minimum block size (>= 1): 2
[INPUT] Enter target compression (0-1, 0 to disable): 0.5
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes4.jpg
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/
4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/res
ult/tes4.gif
```

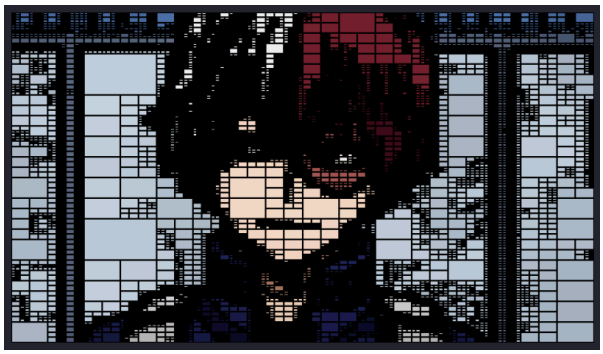
```
[OUTPUT] [THRESHOLD = 200] Compression: 88.7735%
[INFO] GIF saved successfully to: /Users/adndax/Documents
/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tuci
l/test/result/tes4.gif
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4
thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/resu
lt/tes4.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 337 ms
[OUTPUT] Original image size   : 84.7607 KB
[OUTPUT] Compressed image size : 9.51562 KB
[OUTPUT] Compression percentage: 88.7735%
[OUTPUT] Tree depth            : 9
[OUTPUT] Total nodes           : 305
[OUTPUT] Output image path     : /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes4.jpg
```

```
==========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
==========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academi
c/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/t
es5.JPG
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 4
[INPUT] Enter threshold (>= 0): 150
[INPUT] Enter minimum block size (>= 1): 8
[INPUT] Enter target compression (0-1, 0 to disable): 0.2
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes5.JPG
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/
4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/res
ult/tes5.gif
```

```
[OUTPUT] [THRESHOLD = 150] Compression: 68.8833%
[INFO] GIF saved successfully to: /Users/adndax/Documents
/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tuci
l/test/result/tes5.gif
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4
thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/resu
lt/tes5.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 193 ms
[OUTPUT] Original image size   : 29.752 KB
[OUTPUT] Compressed image size : 9.25781 KB
[OUTPUT] Compression percentage: 68.8833%
[OUTPUT] Tree depth            : 1
[OUTPUT] Total nodes           : 1
[OUTPUT] Output image path     : /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes5.JPG
```

```
==========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
==========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academi
c/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/t
es6.JPG
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 5
[INPUT] Enter threshold (>= 0): 0.1
[INPUT] Enter minimum block size (>= 1): 2
[INPUT] Enter target compression (0-1, 0 to disable): 0
[INPUT] Draw outline? (1 = yes, 0 = no): 1
[INPUT] Enter output image path: /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes6.JPG
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/
4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/res
ult/tes6.gif
[INFO] GIF saved successfully to: /Users/adndax/Documents
/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tuci
l/test/result/tes6.gif
```

```
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4
thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/resu
lt/tes6.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 472 ms
[OUTPUT] Original image size   : 32.7998 KB
[OUTPUT] Compressed image size : 38.2061 KB
[OUTPUT] Compression percentage: -16.4826%
[OUTPUT] Tree depth            : 9
[OUTPUT] Total nodes           : 37193
[OUTPUT] Output image path     : /Users/adndax/Documents/
```



```
==========================================
          Welcome to QuaQua Image
       Quadtree -- Divide and Conquer
==========================================

[INPUT] Enter image path: /Users/adndax/Documents/Academi
c/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/t
es7.jpg
[INPUT] Choose error metric (1-5):
   1. Variance
   2. MAD
   3. Max Pixel Difference
   4. Entropy
   5. SSIM
[INPUT] Your choice: 1
[INPUT] Enter threshold (>= 0): 999
[INPUT] Enter minimum block size (>= 1): 16
[INPUT] Enter target compression (0-1, 0 to disable): 0
[INPUT] Draw outline? (1 = yes, 0 = no): 0
[INPUT] Enter output image path: /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes7.jpg
[INPUT] Save as GIF? (1 = yes, 0 = no): 1
[INPUT] Enter GIF path: /Users/adndax/Documents/Academic/
4thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/res
ult/tes7.gif
[INFO] GIF saved successfully to: /Users/adndax/Documents
/Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tuci
l/test/result/tes7.gif
```

```
[OUTPUT] GIF saved at: /Users/adndax/Documents/Academic/4
thSemester/Strategi Algoritma/Tugas Kecil/Tucil/test/resu
lt/tes7.gif

[OUTPUT] ========= COMPRESSION REPORT =========
[OUTPUT] Execution time        : 8244 ms
[OUTPUT] Original image size   : 1217.19 KB
[OUTPUT] Compressed image size : 214.353 KB
[OUTPUT] Compression percentage: 82.3895%
[OUTPUT] Tree depth            : 9
[OUTPUT] Total nodes           : 16217
[OUTPUT] Output image path     : /Users/adndax/Documents/
Academic/4thSemester/Strategi Algoritma/Tugas Kecil/Tucil
/test/result/tes7.jpg
```


4. Hasil Analisis

Algoritma kompresi gambar yang digunakan dalam program ini merupakan implementasi metode *Divide and Conquer* berbasis struktur data Quadtree. Secara umum, algoritma bekerja dengan membagi gambar ke dalam empat kuadran secara rekursif. Proses pembagian dilakukan hingga tiap blok memiliki tingkat keseragaman warna yang cukup, yang diukur berdasarkan nilai ambang batas atau *threshold* dan fungsi error yang dipilih seperti variance, MAD, SSIM, dan sebagainya.

Dari hasil pengujian, algoritma ini menunjukkan performa yang cukup baik dalam hal efisiensi waktu. Hal ini dikarenakan algoritma tidak perlu memproses setiap piksel secara langsung seperti pada pendekatan *brute-force*, melainkan hanya menghitung nilai rata-rata dan error dalam satu blok sebelum memutuskan apakah perlu dibagi lagi atau tidak. Dengan demikian, kompleksitas rata-rata dari algoritma ini dapat dikategorikan sebagai O(n log n), tergantung pada struktur warna dalam gambar dan parameter input

Namun demikian, perlu diperhatikan bahwa hasil kompresi tidak selalu menghasilkan ukuran file yang lebih kecil dari gambar aslinya, terutama jika gambar awal sudah dikompresi menggunakan format seperti JPEG. Dalam beberapa kasus, ukuran file hasil kompresi justru dapat menjadi lebih besar, karena metode ini lebih mengutamakan representasi spasial dan keseragaman warna dibanding efisiensi penyimpanan seperti pada algoritma transformasi frekuensi (misalnya DCT pada JPEG).

Secara keseluruhan, algoritma ini sangat cocok untuk gambar-gambar dengan area warna yang luas dan seragam, serta memiliki keunggulan dari sisi kecepatan dan visualisasi proses kompresi itu sendiri. Penerapannya juga fleksibel dan dapat disesuaikan melalui parameter *threshold*, ukuran blok minimum, serta fungsi error yang digunakan.

5. Implementasi Bonus

5.1. Bonus 1: Implementasi Persentasi Kompresi

Implementasi presentasi kompresi memungkinkan pengguna untuk menentukan target persentase kompresi secara langsung, misalnya 0.4 untuk 40% atau 0.7 untuk 70%. Program akan secara otomatis mencari nilai *threshold* terbaik dengan melakukan iterasi bertingkat dari nilai *threshold* kecil ke besar. Pada setiap iterasi, program menghitung ukuran file hasil kompresi dan membandingkannya dengan ukuran file asli. Apabila target kompresi tercapai, proses dihentikan dan hasil terbaik disimpan.

5.2. Bonus 2: Implementasi Structural Similarity Index (SSIM)

SSIM merupakan salah satu metode evaluasi kualitas gambar terkompresi yang akurat dibandingkan metode lainnya. Dalam bonus ini, SSIM digunakan sebagai salah satu opsi metric error untuk menentukan apakah sebuah blok gambar perlu dibagi lebih lanjut. Implementasi SSIM mempertimbangkan aspek luminance, contrast, dan struktur, sehingga menghasilkan hasil kompresi yang tetap mempertahankan kualitas visual yang baik meskipun terjadi pengurangan ukuran data. SSIM diintegrasikan ke dalam pemilihan metric error bersama dengan metode lain seperti variance, MAD, MPD, dan entropy.

5.3. Bonus 3: Output GIF

Program ini dilengkapi dengan kemampuan menyimpan proses kompresi gambar dalam bentuk animasi GIF. Pada dasarnya, penggunaan output GIF adalah untuk menampilkan bagaimana gambar secara bertahap dikompresi menggunakan metode Quadtree. Namun, implementasi kode saat ini belum sepenuhnya mencerminkan proses pembentukan pohon secara divide-and-conquer.

GIF yang dihasilkan mengambil snapshot berdasarkan kedalaman pohon (depth), bukan berdasarkan urutan rekursif pembentukan node. Akibatnya, animasi terlihat seperti gambar langsung muncul dari warna hitam menjadi penuh secara bertahap per level kedalaman, tetapi tidak secara visual menunjukkan bagaimana gambar dibagi menjadi 4 bagian secara rekursif seperti yang terjadi dalam proses divide and conquer.

Walaupun begitu, fitur ini tetap memberikan gambaran umum tentang kedalaman kompresi dan hasil akhirnya. Untuk membuatnya lebih representatif terhadap proses algoritmanya, ke

depannya bisa dikembangkan dengan traversal per node (preorder) dan menyimpan kondisi gambar setiap kali satu node selesai diproses.

6. Pranala Repository

https://github.com/adndax/Tucil2_13523071

7. Lampiran

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	✓	
2	Program berhasil dijalankan	✓	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4	Mengimplementasi seluruh metode perhitungan error wajib	✓	
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan	✓	
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	
9	Program dibuat oleh saya sendiri	✓	

8. Daftar Pustaka

[1] R. Munir, *Algoritma Divide and Conquer (Bagian 1)*, 11 April 2025. [Daring]. Tersedia: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf