

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma Semester II tahun 2024/2025

Kompresi Gambar Dengan Metode Quadtree



Nicholas Andhika Lucas
13523014

**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025**

Daftar Isi

Daftar Isi.....	2
Pendahuluan.....	3
Deskripsi Masalah.....	3
Algoritma Divide and Conquer.....	5
Penjelasan Algoritma.....	5
Langkah Divide.....	5
Source Program.....	6
Quadtree.java.....	6
QuadtreeNode.java.....	9
ErrorMeasurement.java.....	11
IOHandler.java.....	14
Main.java.....	18
Test Program.....	20
Metode Variance.....	20
Metode Variance.....	20
Metode Mean Absolute Deviation (MAD).....	27
Metode Mean Absolute Deviation (MAD).....	27
Metode Max Pixel Difference (MPD).....	31
Metode Max Pixel Difference (MPD).....	31
Metode Entropy.....	37
Metode Entropy.....	37
Lampiran.....	41

Pendahuluan

Deskripsi Masalah

Gambar yang memiliki ukuran besar dapat memakan memori yang besar dan memboroskan tempat penyimpanan. Sedangkan, gambar tidak setiap saat dibutuhkan untuk dilihat dalam kualitas tertinggi dan ukuran terbesar. Oleh karena itu, hal ini berpotensi mengganggu performa dan kemampuan *processing*. Berdasarkan masalah tersebut, solusi yang hadir adalah *image compression*. *Image compression* memiliki tujuan untuk mengurangi ukuran dari gambarnya sendiri tanpa atau meminimalisir pengurangan kualitas. Terutama untuk gambar-gambar yang berukuran besar, banyak *pixel* yang secara individual memiliki identitas warna yang mirip dengan tetangganya, sehingga secara intuitif dapat disatukan karena tidak akan menyebabkan perubahan kualitas secara signifikan.

Quadtree adalah salah satu struktur data yang dimanfaatkan untuk *image compression*. Quadtree memiliki struktur yang sama seperti struktur pohon (*tree*), namun memiliki 4 anak (*child*) dibandingkan 2 pada umumnya – jika berbicara tentang *binary tree* -. Dalam konteks *image compression*, quadtree dapat membagi gambar menjadi blok-blok yang lebih kecil berdasarkan keseragaman warna atau intensitas pi. 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.

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.

Pada tugas kecil kali ini, kami ditugaskan untuk mengimplementasikan struktur data quadtree dan memanfaatkan algoritma *Divide and Conquer* untuk melakukan *image compression*.

Algoritma *Divide and Conquer*

Penjelasan Algoritma

Secara garis besar, algoritma *divide and conquer* merujuk pada pendekatan penyelesaian persoalan dengan cara membagi persoalan menjadi beberapa upa-persoalan yang lebih kecil dan mirip, memecahkan tiap upa-persoalan secara rekursif (atau langsung jika persoalannya kecil), dan menggabungkan solusi tiap upa-persoalan menjadi solusi persoalan awal. Berdasarkan definisi ini, langkah dalam penyelesaian persoalan dengan algoritma *divide and conquer* dapat dibagi menjadi 3, yaitu *divide*, *conquer*, dan *combine*.

Untuk implementasi dalam persoalan tugas kecil ini, ide dari algoritmanya adalah membagi suatu gambar menjadi sub-blok persegi, memanfaatkan struktur dari Quadtree yang memiliki 4 *child node*. Tiap sub-blok ini memiliki warna rata-rata yang diambil dari bagian gambar asli. Apabila nilai variansi warna tersebut masih di atas ambang batas variansi yang ditentukan pengguna, blok akan dipecah lagi menjadi 4 upa-blok persegi. Ketika quadtree sudah tidak bisa lagi dibagi, maka warna tiap blok akan dinormalisasi (untuk menggambarkan sejumlah blok yang direpresentasikannya), dan akan dibentuk kembali gambar berdasarkan struktur quadtree.

Langkah Divide

Bagi blok menjadi 4 upa-blok secara rekursif, terdiri dari blok kiri atas, kanan atas, kiri bawah, dan kanan bawah; apabila memenuhi 2 kondisi ini:

1. Nilai variansi warna blok berada di atas ambang batas yang ditentukan
2. Ukuran blok lebih besar dari *minimum block size* yang ditentukan, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari *minimum block size*.

*Tidak dicantumkan pseudocode yang menjelaskan cara kerja fungsi *split* untuk membagi-bagi blok menjadi sub-blok. Lebih menonjolkan logika dari syarat untuk divide dan pemanggilan rekursifnya.

```
function shouldSplit(BufferedImage img, QuadtreeNode node) → boolean
{ Menentukan apakah suatu blok harus dibagi menjadi upablok lagi atau tidak }
KAMUS LOKAL
area: Rectangle
childBlockArea: integer
error: double
```

ALGORITMA

```

area ← node.getArea()
childBlockArea ← area.width * area.height

{ Kondisi 1: Cek apabila blok setelah dibagi akan lebih kecil dari
minBlockSize }
if (childBlockArea < minBlockSize) then
→ false

{ Kondisi 2: Cek perbandingan nilai variansi terhadap batas ambang }

error ← calculateError(img, area)
→ error > threshold

```

Kode 1. Fungsi shouldSplit()

```

procedure buildRecursive(node: QuadtreeNode)
{ Membangun quadtree secara rekursif dengan membagi node yang memenuhi
kriteria }
KAMUS LOKAL
    child: QuadtreeNode

ALGORITMA
    if (shouldSplit(img, node)) then
        node.split(img)
        for each child in node.getChildren() do
            buildRecursive(child)

procedure buildQuadtree()
{ Fungsi utama untuk memulai konstruksi quadtree dari root }

ALGORITMA
    buildRecursive(root)

```

Kode 2. Fungsi buildRecursive dan buildQuadtree

Langkah Conquer

Tahap *divide* berakhir jika setiap cabang node atau blok tidak lagi bisa dibagi. Setelah itu, solusi yang diambil untuk tiap upa-node atau upa-blok adalah menormalisasi warnanya dengan cara menghitung nilai RGB rata-rata dari bagian gambar, kemudian memberikan warna rata-rata tersebut pada upa-blok. Dengan demikian, terbentuk kompresi dari gambarnya – mewarnakan tetangga piksel yang mirip dengan warna yang sama, mengurangi memori ekstra untuk menyimpan nilai RGB secara individual –.

Dalam konteks program saya, tidak ada fungsi terpisah yang secara eksplisit mengimplementasikan solusi dalam *conquer*. Warna di sini merupakan atribut dari Node,

maka setiap kali blok dibagi menjadi 4 upa-blok, akan dihitung kembali warna blok tersebut berdasarkan warna rata-rata bagian gambar yang dihampirinya.

*Tidak ditunjukkan juga implementasi fungsi `calculateAvgColor` untuk menghitung warna rata-rata. Anggap saja seperti OOP, tidak perlu tahu prosedurnya tapi tau fungsinya :D

```
procedure split(img: BufferedImage)
{ Membagi node menjadi 4 anak dengan ukuran yang sesuai }
KAMUS LOKAL
    topLeft, topRight, bottomLeft, bottomRight: Rectangle
ALGORITMA
    children ← new QuadtreeNode[4]
    { Perhitungan ukuran blok . . . }

    { Blok Kiri Atas }
    topLeft ← new Rectangle(area.x, area.y, remainingWidth, remainingHeight)
    children[0] ← new QuadtreeNode(topLeft, calculateAvgColor(img, topLeft))

    { Blok Kanan Atas }
    topRight ← new Rectangle(area.x + halfWidth, area.y, remainingWidth,
remainingHeight)
    children[1] ← new QuadtreeNode(topRight, calculateAvgColor(img, topRight))

    { Diulang untuk blok selanjutnya }
```

Kode 3. Tahap Conquer dari Fungsi Split

Langkah Combine

Setelah setiap upa-blok diselesaikan pada langkah *conquer* sebelumnya, langkah *combine* akan menggabungkan seluruh upa-blok yang sudah terstruktur sedemikian rupa dalam Quadtree agar menjadi 1 gambar utuh kembali.

```
function reconstruct() → BufferedImage
{ Membangun gambar hasil kompresi dari struktur quadtree }
KAMUS LOKAL
    output: BufferedImage
    g: Graphics2D

ALGORITMA
    { Inisialisasi gambar output }
```

```

output ← new BufferedImage(img.width, img.height, TYPE_INT_RGB)
g ← output.createGraphics()

{ Render pohon mulai dari root }
renderQuadtree(g, root)

{ Finalisasi }
g.dispose()
→ output

procedure renderQuadtree(input g: Graphics2D, input node: QuadtreeNode, output
Quadtree)
{ Merender node quadtree ke gambar output secara rekursif }

KAMUS LOKAL
area: Rectangle
child: QuadtreeNode

ALGORITMA
if (node.isLeaf()) then
    { BASE CASE: Render blok yang tidak memiliki anak lagi }
    g.setColor(node.getAvgColor())
    area ← node.getArea()
    g.fillRect(area.x, area.y, area.width, area.height)

else
    { RECURSIVE CASE: Menelusuri upa-blok }
    for each child in node.getChildren() do
        renderQuadtree(g, child)

```

Source Program

Program ini menggunakan bahasa pemrograman Java.

Quadtree.java

Berfungsi untuk mendefinisikan kelas Quadtree yang dimanfaatkan untuk melakukan kompresi pada gambar. Terdiri dari atribut utama method, threshold, dan minBlockSize beserta atribut tambahan image dan root node. Metode yang diimplementasikan berupa fungsi untuk melakukan perhitungan *error*, melakukan pembagian tree menjadi 4 node, dan konstruksi kembali tree menjadi gambar.

```
import java.awt.image.BufferedImage;
import java.awt.Rectangle;
import java.awt.Graphics2D;

public class Quadtree {
    /* Attributes */
    private ErrorMeasurement.Method method;
    private double threshold;
    private int minBlockSize;
    private BufferedImage img;
    private QuadtreeNode root;

    /* Constructor */
    public Quadtree(ErrorMeasurement.Method method, double threshold, int minBlockSize, BufferedImage img) {
        this.method = method;
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.img = img;
        Rectangle rectangle = new Rectangle(0, 0, img.getWidth(), img.getHeight());
        this.root = new QuadtreeNode(rectangle, QuadtreeNode.calculateAvgColor(img, rectangle));
    }

    /* Getters */
    public double getThreshold() {
        return threshold;
    }

    public int getMinBlockSize() {
        return minBlockSize;
    }

    public BufferedImage getImg() {
        return img;
    }
}
```

```

    }

    public QuadtreeNode getRoot() {
        return root;
    }

    public int getDepth(QuadtreeNode node) {
        if (node == null || node.isLeaf()){
            return 1;
        }
        int maxDepth = 0;
        if (node.getChildren() != null){
            for (QuadtreeNode child : node.getChildren()){
                maxDepth = Math.max(maxDepth, getDepth(child));
            }
        }
        return maxDepth + 1;
    }

    public int getNodeCount(QuadtreeNode node) {
        if (node == null){
            return 0;
        }
        int count = 1;
        if (!node.isLeaf() && node.getChildren() != null){
            for (QuadtreeNode child : node.getChildren()){
                if (child != null) count += getNodeCount(child);
            }
        }
        return count;
    }

    /* Methods */
    /* Calculate value of error based on the chosen method */
    private double calculateError(BufferedImage image, Rectangle area) {
        ErrorMeasurement em = new ErrorMeasurement();
        switch (this.method) {
            case VARIANCE: return em.variance(image, area);
            case MAD: return em.mad(image, area);
            case MPD: return em.mpd(image, area);
            case ENTROPY: return em.entropy(image, area);
            default: throw new IllegalArgumentException("Unknown error measurement
method.");
        }
    }

    /* Determines whether an image pixel should be split further or not */
    public boolean shouldSplit(BufferedImage img, QuadtreeNode node){
        Rectangle area = node.getArea();
        int childBlockArea = area.width * area.height;

        // Condition 1: Stop if child block would be smaller than minBlockSize

```

```

        if (childBlockArea < minBlockSize){
            return false;
        }

        // Condition 2: Stop if error is not above threshold
        double error = calculateError(img, area);
        return error > threshold;
    }

    /* Recursive function for buildQuadtree */
    private void buildRecursive(QuadtreeNode node){
        if (shouldSplit(img, node)){
            node.split(img);
            for (QuadtreeNode child : node.getChildren()){
                buildRecursive(child);
            }
        }
    }

    /* Actual function to build quadtree, starting from the root */
    public void buildQuadtree() {
        buildRecursive(root);
    }

    /* Reconstruct compressed image from quadtree */
    public BufferedImage reconstruct(){
        // Create a new image, same dimensions with the input image
        BufferedImage output = new BufferedImage(
            img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_RGB
        );
        Graphics2D g = output.createGraphics();

        // Combine quadtree with recursive function. Graphics2D draws on the output
        image.
        renderQuadtree(g, root);
        g.dispose();
        return output;
    }

    private void renderQuadtree(Graphics2D g, QuadtreeNode node){
        // Base
        if (node.isLeaf()){
            g.setColor(node.getAvgColor());
            Rectangle area = node.getArea();
            g.fillRect(area.x, area.y, area.width, area.height);
        }
        // Recursion
        else {
            for (QuadtreeNode child : node.getChildren()){
                renderQuadtree(g, child);
            }
        }
    }
}

```

```
    }  
}
```

QuadtreeNode.java

Berfungsi untuk mendefinisikan kelas Node yang dimanfaatkan untuk Quadtree. Terdiri dari atribut utama area sebagai Rectangle untuk mendapatkan nilai panjang dan posisinya, average color dari node/area gambar tersebut, dan child node sebanyak 4 yang diurutkan berdasarkan *top-left*, *top-right*, *bottom-left*, dan *bottom-right*. Metode yang diimplementasikan mencakup perhitungan warna rata-rata pada blok gambar dan memisahkan suatu node Quadtree menjadi 4 child node.

```
import java.awt.*;  
import java.awt.image.BufferedImage;  
  
public class QuadtreeNode {  
    /* Attributes */  
    private Rectangle area; // x, y, width, height  
    private Color avgColor;  
    private QuadtreeNode[] children; // top-left, top-right, bottom-left,  
    bottom-right  
  
    /* Constructor */  
    public QuadtreeNode(Rectangle area, Color avgColor){  
        this.area = area;  
        this.avgColor = avgColor;  
        this.children = null;  
    }  
  
    /* Getters */  
    public Rectangle getArea(){  
        return this.area;  
    }  
  
    public Color getAvgColor(){  
        return this.avgColor;  
    }  
  
    public QuadtreeNode[] getChildren(){  
        return this.children;  
    }  
  
    public boolean isLeaf(){  
        return this.children == null;  
    }
```

```

/*Functions */
/* Calculate the average color of a node (area of pixel) */
public static Color calculateAvgColor(BufferedImage img, Rectangle area){
    int red = 0;
    int green = 0;
    int blue = 0;
    int pixelCount = 0;

    // Iterate box of pixels in the image to sum RGB values
    for (int y = area.y; y < area.y + area.height; y++){
        for (int x = area.x; x < area.x + area.width; x++){
            int RGB = img.getRGB(x, y);
            Color color = new Color(RGB);
            red += color.getRed();
            green += color.getGreen();
            blue += color.getBlue();
            pixelCount++;
        }
    }

    // Edge case for invalid iteration
    if (pixelCount == 0){
        return new Color(0, 0, 0);
    }

    // Create new Color object with average RGB values and return
    int avgRed = (int) (red / pixelCount);
    int avgGreen = (int) (green / pixelCount);
    int avgBlue = (int) (blue / pixelCount);

    return new Color(avgRed, avgGreen, avgBlue);
}

/* Splits given image section into 4 boxes. */
public void split(BufferedImage img){
    this.children = new QuadtreeNode[4];
    int halfWidth = area.width / 2;
    int halfHeight = area.height / 2;
    int remainingWidth = area.width - halfWidth;
    int remainingHeight = area.height - halfHeight;

    children = new QuadtreeNode[4];
    // Top-Left
    Rectangle topLeft = new Rectangle(area.x, area.y, remainingWidth,
remainingHeight);
    children[0] = new QuadtreeNode(topLeft, QuadtreeNode.calculateAvgColor(img,
topLeft));

    // Top-Right
    Rectangle topRight = new Rectangle(area.x + halfWidth, area.y,
remainingWidth, remainingHeight);
}

```

```

        children[1] = new QuadtreeNode(topRight,
QuadtreeNode.calculateAvgColor(img, topRight));

        // Bottom-Left
        Rectangle bottomLeft = new Rectangle(area.x, area.y + halfHeight,
remainingWidth, remainingHeight);
        children[2] = new QuadtreeNode(bottomLeft,
QuadtreeNode.calculateAvgColor(img, bottomLeft));

        // Bottom-Right
        Rectangle bottomRight = new Rectangle(area.x + halfWidth, area.y +
halfHeight, remainingWidth, remainingHeight);
        children[3] = new QuadtreeNode(bottomRight,
QuadtreeNode.calculateAvgColor(img, bottomRight));
    }
}

```

ErrorMeasurement.java

Berfungsi untuk mendefinisikan karakteristik/perhitungan dari metode-metode pengukuran error yang dipilih pengguna. Metode yang didefinisikan mencakup *variance*, *MAD*, *MPD*, dan *entropy*.

```

import java.awt.*;
import java.awt.image.BufferedImage;
public class ErrorMeasurement {
    public enum Method {
        VARIANCE, MAD, MPD, ENTROPY
    }

    /* 1. Variance ( $\sigma^2$ ) */
    public double variance(BufferedImage img, Rectangle area) {
        // Edge case for empty areas
        if (area.width <= 0 || area.height <= 0) {
            return 0; // Return 0 for empty areas
        }
        Color avgRGB = QuadtreeNode.calculateAvgColor(img, area);
        double varianceR = 0;
        double varianceG = 0;
        double varianceB = 0;
        int pixelCount = 0;

        /* Count variance for individual R, G, and B values */
        for (int y = area.y; y < area.y + area.height; y++) {
            for (int x = area.x; x < area.x + area.width; x++) {

```

```

        if (x >= img.getWidth() || y >= img.getHeight()) continue;

        Color pixelRGB = new Color(img.getRGB(x, y));
        varianceR += Math.pow(pixelRGB.getRed() - avgRGB.getRed(), 2);
        varianceG += Math.pow(pixelRGB.getGreen() - avgRGB.getGreen(), 2);
        varianceB += Math.pow(pixelRGB.getBlue() - avgRGB.getBlue(), 2);
        pixelCount++;
    }
}
/* Edge case if no valid pixels were iterated */
if (pixelCount == 0) return 0;

/* Return the final variance of RGB */
return (varianceR + varianceG + varianceB) / (3 * pixelCount);
}

/* 2. Mean Absolute Deviation (MAD) */
public double mad(BufferedImage img, Rectangle area) {
    // Edge case for empty areas
    if (area.width <= 0 || area.height <= 0) {
        return 0; // Return 0 for empty areas
    }
    Color avgRGB = QuadtreeNode.calculateAvgColor(img, area);
    double madR = 0;
    double madG = 0;
    double madB = 0;
    int pixelCount = 0;

    /* Count MAD for individual R, G, and B values */
    for (int y = area.y; y < area.y + area.height; y++) {
        for (int x = area.x; x < area.x + area.width; x++) {

            if (x >= img.getWidth() || y >= img.getHeight()) continue;

            Color pixelRGB = new Color(img.getRGB(x, y));
            madR += Math.abs(pixelRGB.getRed() - avgRGB.getRed());
            madG += Math.abs(pixelRGB.getGreen() - avgRGB.getGreen());
            madB += Math.abs(pixelRGB.getBlue() - avgRGB.getBlue());
            pixelCount++;
        }
    }
    /* Edge case if no valid pixels were iterated */
    if (pixelCount == 0) return 0;

    /* Return the final MAD of RGB */
    return (madR + madG + madB) / (3 * pixelCount);
}

/* 3. Maximum Pixel Difference (MPD) */
public double mpd(BufferedImage img, Rectangle area){
    // Edge case for empty areas

```

```

if (area.width <= 0 || area.height <= 0) {
    return 0; // Return 0 for empty areas
}
double maxR = 0;
double minR = 0;
double maxG = 0;
double minG = 0;
double maxB = 0;
double minB = 0;
int pixelCount = 0;

/* Update the max and min value for each color channel */
for (int y = area.y; y < area.y + area.height; y++){
    for (int x = area.x; x < area.x + area.width; x++){
        if (x >= img.getWidth() || y >= img.getHeight()) continue;

        Color pixelRGB = new Color(img.getRGB(x, y));

        if (pixelRGB.getRed() > maxR) maxR = pixelRGB.getRed();
        if (pixelRGB.getRed() < minR) minR = pixelRGB.getRed();
        if (pixelRGB.getGreen() > maxG) maxG = pixelRGB.getGreen();
        if (pixelRGB.getGreen() < minG) minG = pixelRGB.getGreen();
        if (pixelRGB.getBlue() > maxB) maxB = pixelRGB.getBlue();
        if (pixelRGB.getBlue() < minB) minB = pixelRGB.getBlue();
        pixelCount++;
    }
}
/* Edge case if no valid pixels were iterated */
if (pixelCount == 0) return 0;

/* Return the difference in RGB */
return ((maxR - minR) + (maxG - minG) + (maxB - minB)) / 3;
}

/* 4. Entropy */
public double entropy(BufferedImage img, Rectangle area){
    // Edge case for empty areas
    if (area.width <= 0 || area.height <= 0) {
        return 0; // Return 0 for empty areas
    }
    int pixelCount = 0;

    // Plot the frequency of each color channel value in histogram
    int[][] histogram = new int[3][256]; // R, G, B

    for (int y = area.y; y < area.y + area.height; y++){
        for (int x = area.x; x < area.x + area.width; x++){
            if (x >= img.getWidth() || y >= img.getHeight()) continue;

            Color pixelRGB = new Color(img.getRGB(x, y));
            histogram[0][pixelRGB.getRed()]++;
            histogram[1][pixelRGB.getGreen()]++;
        }
    }
}

```

```

        histogram[2][pixelRGB.getBlue()]++;
        pixelCount++;
    }
}

/* Edge case if no valid pixels were iterated */
if (pixelCount == 0) return 0;

// Count entropy value per each RGB channel
double[] channelEntropy = new double[3];
for (int i = 0; i < 3; i++){ // RGB
    double entropy = 0;
    for (int j = 0; j < 256; j++){ // 0 - 255 channel values
        if (histogram[i][j] > 0){
            double probability = (double) histogram[i][j] / pixelCount;
            entropy -= probability * Math.log(probability) / Math.log(2);
        }
    }
    channelEntropy[i] = entropy;
}
// Return the final entropy for RGB
return (channelEntropy[0] + channelEntropy[1] + channelEntropy[2]) / 3;
}
}

```

IOHandler.java

Berfungsi untuk menciptakan fungsi-fungsi yang berhubungan dengan input/output pada file Main.java utama (menghindari source code kotor). Setiap fungsi memanfaatkan *try catch* untuk menghindari keluarnya program akibat input yang tidak sesuai.

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.Scanner;
import javax.imageio.ImageIO;
import java.util.AbstractMap.SimpleEntry;
import java.text.DecimalFormat;

public class IOHandler {
    /* Attributes */
    private Scanner scanner;

    /* Constructor */
    public IOHandler() {
        this.scanner = new Scanner(System.in);
    }
}

```

```

}

/* Methods */
/* 1. Input absolute image path */
public SimpleEntry<BufferedImage, String> inputImage(){
    while (true){
        System.out.print("\nMasukkan alamat absolut gambar yang akan dikompresi:");
    });
    String path = scanner.nextLine();
    try {
        BufferedImage img = ImageIO.read(new File(path));
        if (img == null) {
            System.out.println("Gambar tidak ditemukan pada alamat tersebut. Silahkan coba lagi!");
        } else{
            return new SimpleEntry<>(img, path);
        }
    } catch (Exception e) {
        System.out.println("Terjadi kesalahan saat membaca gambar pada path tersebut. Silahkan coba lagi!");
    }
}
}

/* 2. Input error measurement method */
public ErrorMeasurement.Method inputErrorMethod(){
    System.out.println("Metode Pengukuran Error:\n1. Variance\n2. Mean Absolute Deviation (MAD)\n3. Maximum Pixel Difference (MPD)\n4. Entropy\n");
    System.out.print("Pilih metode pengukuran error (1-4): ");
    while (true){
        try{
            int input = Integer.parseInt(scanner.nextLine());
            if (input < 1 || input > 4) {
                System.out.println("Pilihan metode di luar batas angka 1-4!");
            } else {
                return ErrorMeasurement.Method.values()[input - 1];
            }
        } catch (NumberFormatException e){
            System.out.println("Masukkan pilihan metode yang valid berupa angka 1-4!");
        }
    }
}

/* 3. Input threshold */
public double inputThreshold(ErrorMeasurement.Method method){
    double bottomRange = 0;
    double topRange = 0;
    switch(method){
        case VARIANCE:
            System.out.println("Threshold Metode Variance: (100-1000)");
            bottomRange = 100;

```

```

        topRange = 1000;
        break;
    case MAD:
        System.out.println("Threshold Metode MAD: (5-50)");
        bottomRange = 5;
        topRange = 50;
        break;
    case MPD:
        System.out.println("Threshold Metode MPD: (10-200)");
        bottomRange = 10;
        topRange = 200;
        break;
    case ENTROPY:
        System.out.println("Threshold Metode Entropy: (0.5-5)");
        bottomRange = 0.5;
        topRange = 5;
        break;
    }

    while (true){
        try{
            System.out.print("Masukkan nilai threshold yang di antara batas:");
        );
        double input = Double.parseDouble(scanner.nextLine());
        if (input < bottomRange || input > topRange) {
            System.out.println("Nilai threshold di luar batas threshold!");
        } else {
            return input;
        }
    } catch (NumberFormatException e){
        System.out.println("Masukkan nilai threshold yang valid!");
    }
}
}

/* 4. Input minimum block size */
public int inputMinBlockSize() {
    while(true){
        try {
            System.out.print("Masukkan ukuran blok minimum dalam integer");
(Contoh: 800 untuk blok 40x20): ");
            int input = Integer.parseInt(scanner.nextLine());
            if (input < 1) {
                System.out.println("Ukuran blok minimum tidak valid! Silahkan coba lagi.");
            } else {
                return input;
            }
        } catch (NumberFormatException e) {
            System.out.println("Masukkan ukuran blok minimum yang valid!");
        }
    }
}

```

```

}

/* 6. Input result absolute path */
public String inputOutputPath() {
    while (true) {
        try {
            System.out.print("Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: ");
            String path = scanner.nextLine().trim();
            File file = new File(path);
            String parent = file.getParent();

            if (parent == null || parent.isEmpty()) {
                throw new IllegalArgumentException("Path absolut tidak valid! Pastikan input alamat direktori yang lengkap.");
            } else {
                return path;
            }
        } catch (IllegalArgumentException e) {
            System.out.println(e.getMessage());
        } catch (Exception e) {
            System.out.println("Terjadi kesalahan yang tidak terduga: " + e.getMessage());
        }
    }
}

/* 8-13. Output exec time, before and after image size, compression percentage, tree depth and node count */
public void outputStats(long startTime, long endTime, String inputPath, File outputFile, Quadtree quadtree) {
    double executionTime = (endTime - startTime) / 1_000_000_000;
    long originalSize = new File(inputPath).length();
    long compressedSize = outputFile.length();

    if (originalSize == 0){
        System.out.println("Ukuran gambar asli adalah 0. Tidak dapat menghitung rasio kompresi.");
        return;
    }

    double compressionRatio = ((double) (originalSize - compressedSize) / originalSize) * 100;
    int depth = quadtree.getDepth(quadtree.getRoot());
    int nodeCount = quadtree.getNodeCount(quadtree.getRoot());

    DecimalFormat df = new DecimalFormat("#.###");
    System.out.println("\nStatistik Kompresi Gambar");
    System.out.println("Waktu eksekusi: " + executionTime + " detik");
    System.out.println("Ukuran gambar sebelum: " + originalSize + " Bytes (" + (originalSize / 1024) + " KB)");
}

```

```

        System.out.println("Ukuran gambar setelah: " + compressedSize + " Bytes (" +
+ (compressedSize / 1024) + " KB)");
        System.out.println("Persentase kompresi: " + df.format(compressionRatio) +
"%");
        System.out.println("Kedalaman pohon: " + depth);
        System.out.println("Banyak simpul pada pohon: " + nodeCount);
    }

/* 14. Output the compressed image at given output path */
public void outputImage(BufferedImage image, String outputPath) {
    try {
        File outputFile = new File(outputPath);
        ImageIO.write(image, "jpg", outputFile);
        System.out.println("Gambar hasil kompresi berhasil disimpan di: " +
outputPath);
    } catch (Exception e) {
        System.out.println("Terjadi kesalahan saat menyimpan gambar hasil
kompresi: " + e.getMessage());
    }
}

public void close(){
    scanner.close();
}
}

```

Main.java

File Main utama. Menggabungkan seluruh fungsi dan menciptakan proses input output kepada pengguna sesuai dengan spesifikasi.

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.AbstractMap.SimpleEntry;

public class Main {
    public static void main(String[] args){
        IOHandler io = new IOHandler();

        /* 1. Input absolute image path */
        SimpleEntry<BufferedImage, String> inputMap= io.inputImage();
        BufferedImage image = inputMap.getKey();
        String inputPath = inputMap.getValue();

        /* 2. Input error measurement method */
        ErrorMeasurement.Method method = io.inputErrorMethod();
    }
}

```

```
/* 3. Input threshold */
double threshold = io.inputThreshold(method);

/* 4. Input minBlockSize */
int minBlockSize = io.inputMinBlockSize();

/* 5. Input Target Compression (bonus) */

/* 6. Input Output Compressed Image Path */
String outputPath = io.inputOutputPath();
File outputFile = new File(outputPath);

/* 7. Input Output GIF Path */

/* 8-14. Output Image and Stats */
long startTime = System.nanoTime();
Quadtree quadtree = new Quadtree(method, threshold, minBlockSize, image);
quadtree.buildQuadtree();
BufferedImage compressedImage = quadtree.reconstruct();
long endTime = System.nanoTime();

io.outputImage(compressedImage, outputPath);
io.outputStats(startTime, endTime, inputPath, outputFile, quadtree);

io.close();
}
```

Test Program

Pada praktikum pertama, alamat IP sebuah perangkat ditentukan secara manual; jika dilakukan seperti ini, alamat tersebut dinamakan **IP statis**. Di jaringan-jaringan yang lebih

Metode *Variance*

Metode <i>Variance</i>	
1	<pre>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\1.jpg Metode Pengukuran Error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Maximum Pixel Difference (MPD) 4. Entropy Pilih metode pengukuran error (1-4): 1 Threshold Metode Variance: (100-1000) Masukkan nilai threshold yang di antara batas: 200 Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 10 Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\1.jpg Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\1.jpg Statistik Kompresi Gambar Waktu eksekusi: 1.0 detik Ukuran gambar sebelum: 944233 Bytes (922 KB) Ukuran gambar setelah: 149252 Bytes (145 KB) Persentase kompresi: 84.19% Kedalaman pohon: 11 Banyak simpul pada pohon: 116077</pre> <p style="text-align: center;"><i>Hasil tangkapan layar.</i></p>



Hasil kompresi gambar.

2

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\2.JPG

Metode Pengukuran Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Maximum Pixel Difference (MPD)
4. Entropy

Pilih metode pengukuran error (1-4): 1

Threshold Metode Variance: (100-1000)

Masukkan nilai threshold yang di antara batas: 100

Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 15

Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\2.jpg

Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\2.jpg

Statistik Kompresi Gambar

Waktu eksekusi: 1.0 detik

Ukuran gambar sebelum: 581287 Bytes (567 KB)

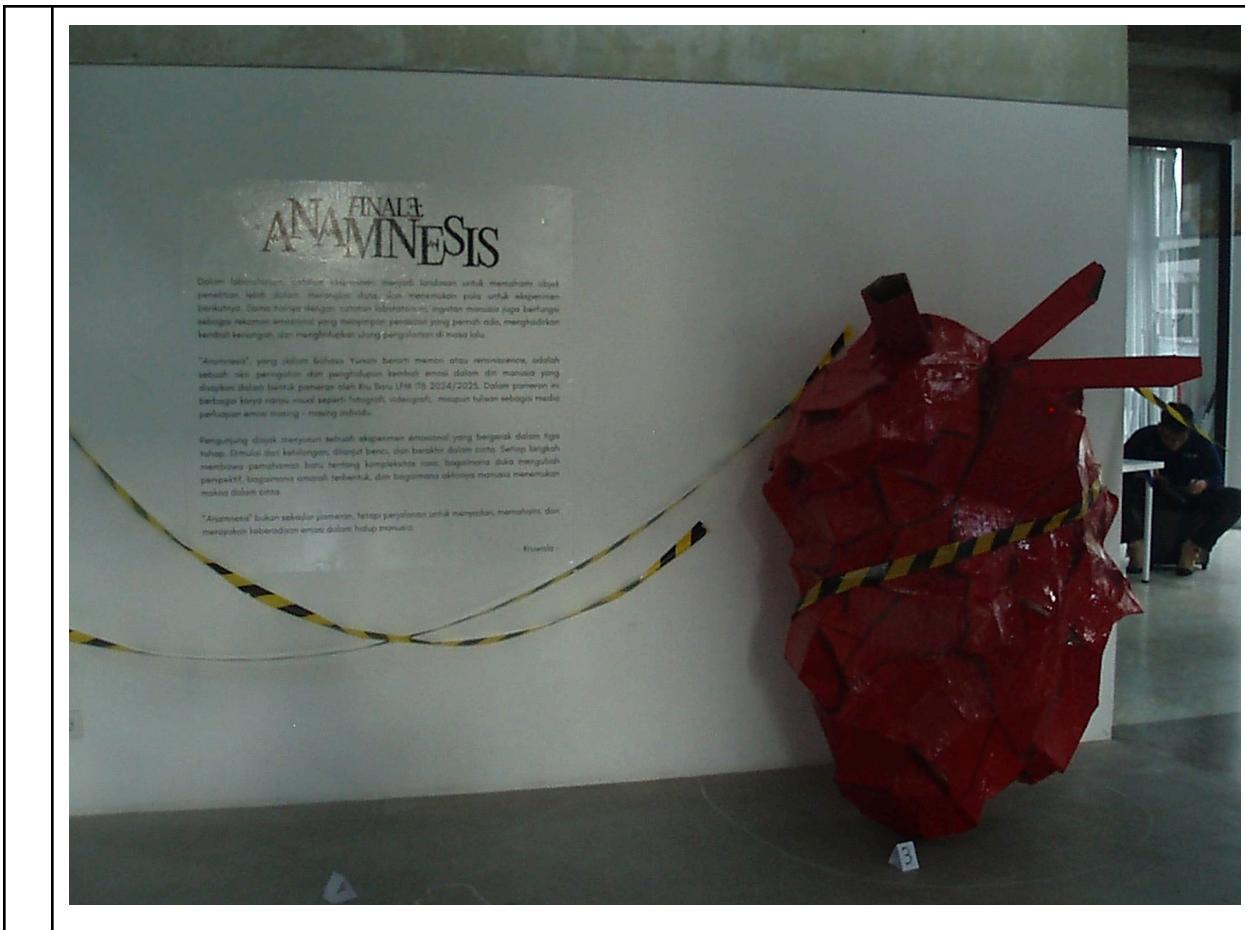
Ukuran gambar setelah: 97626 Bytes (95 KB)

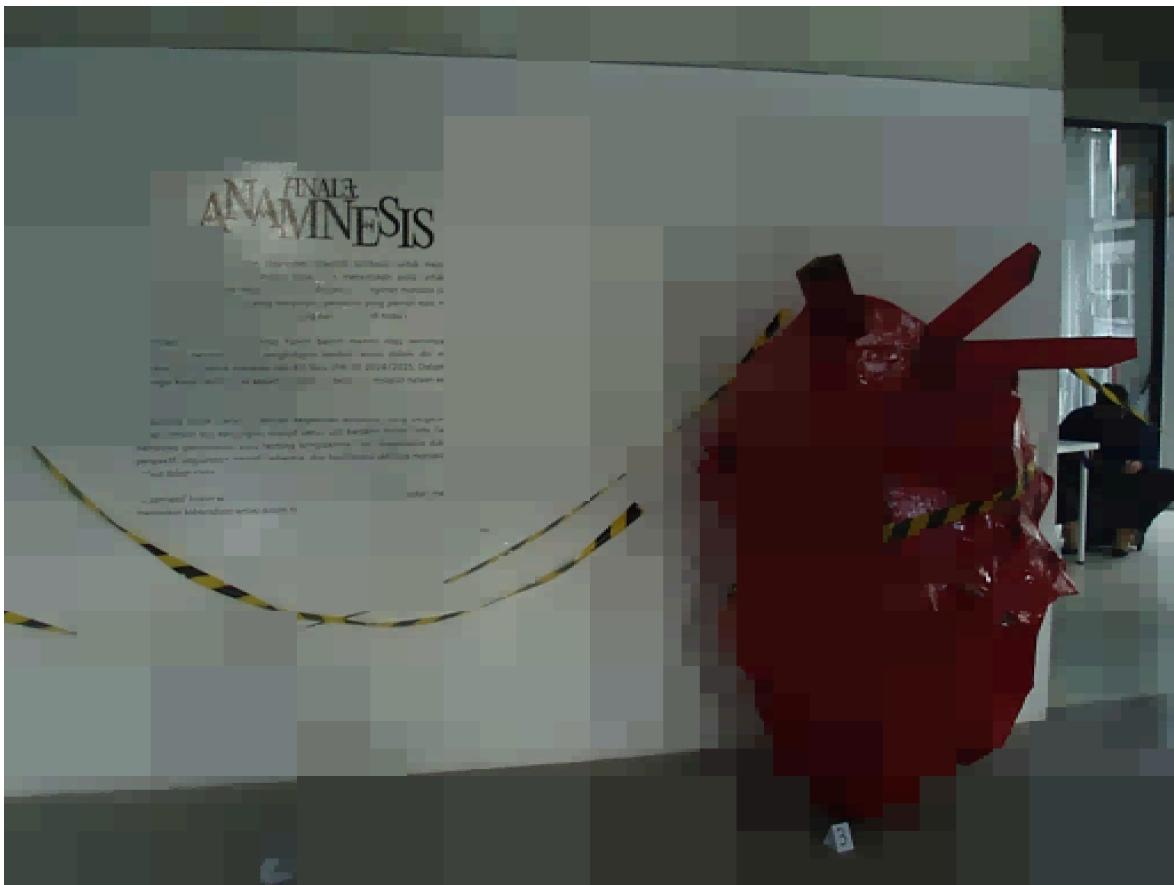
Persentase kompresi: 83,205%

Kedalaman pohon: 10

Banyak simpul pada pohon: 26497

Hasil tangkapan layar.





Hasil kompresi gambar.

3

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\3.jpg
Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Maximum Pixel Difference (MPD)
4. Entropy

Pilih metode pengukuran error (1-4): 1
Threshold Metode Variance: (10-500)
Masukkan nilai threshold yang di antara batas: 10
Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 4
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\3.jpg
Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\3.jpg

Statistik Kompresi Gambar
Waktu eksekusi: 2.0 detik
Ukuran gambar sebelum: 627341 Bytes (612 KB)
Ukuran gambar setelah: 197966 Bytes (193 KB)
Persentase kompresi: 68.444%
Kedalaman pohon: 12
Banyak simpul pada pohon: 886265

Hasil tangkapan layar.





Hasil kompresi gambar.

Metode *Mean Absolute Deviation* (MAD)

Metode <i>Mean Absolute Deviation</i> (MAD)	
1	<p>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\4.jpg Metode Pengukuran Error:</p> <ul style="list-style-type: none">1. Variance2. Mean Absolute Deviation (MAD)3. Maximum Pixel Difference (MPD)4. Entropy <p>Pilih metode pengukuran error (1-4): 2 Threshold Metode MAD: (5-50) Masukkan nilai threshold yang di antara batas: 5 Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 100 Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\4.jpg Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\4.jpg</p> <p>Statistik Kompresi Gambar Waktu eksekusi: 1.0 detik Ukuran gambar sebelum: 1079428 Bytes (1054 KB) Ukuran gambar setelah: 187101 Bytes (182 KB) Persentase kompresi: 82.667% Kedalaman pohon: 9 Banyak simpul pada pohon: 56545</p> <p><i>Hasil tangkapan layar.</i></p>



Hasil kompresi gambar.

2

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\5.jpg

Metode Pengukuran Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Maximum Pixel Difference (MPD)
4. Entropy

Pilih metode pengukuran error (1-4): 2

Threshold Metode MAD: (5-50)

Masukkan nilai threshold yang di antara batas: 20

Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 50

Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\5.jpg
Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\5.jpg

Statistik Kompresi Gambar

waktu eksekusi: 0.0 detik

Ukuran gambar sebelum: 536736 Bytes (524 KB)

Ukuran gambar setelah: 182854 Bytes (178 KB)

Persentase kompresi: 65.932%

Kedalaman pohon: 9

Banyak simpul pada pohon: 39369

Hasil tangkapan layar.





Hasil kompresi gambar.

Metode *Max Pixel Difference* (MPD)

Metode <i>Max Pixel Difference</i> (MPD)	
1	<p>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\6.jpg Metode Pengukuran Error:</p> <ul style="list-style-type: none">1. Variance2. Mean Absolute Deviation (MAD)3. Maximum Pixel Difference (MPD)4. Entropy <p>Pilih metode pengukuran error (1-4): 3 Threshold Metode MPD: (10-200) Masukkan nilai threshold yang di antara batas: 30 Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 10 Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\6.jpg Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\6.jpg</p> <p>Statistik Kompresi Gambar Waktu eksekusi: 0.0 detik Ukuran gambar sebelum: 191432 Bytes (186 KB) Ukuran gambar setelah: 183017 Bytes (178 KB) Persentase kompresi: 4.396% Kedalaman pohon: 10 Banyak simpul pada pohon: 253393</p> <p><i>Hasil tangkapan layar.</i></p>





	<i>Hasil kompresi gambar.</i>
2	<p>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\7.jpg Metode Pengukuran Error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Maximum Pixel Difference (MPD) 4. Entropy</p> <p>Pilih metode pengukuran error (1-4): 3 Threshold Metode MPD: (10-200) Masukkan nilai threshold yang di antara batas: 42 Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 20 Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\7.jpg Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\7.jpg</p> <p>Statistik Kompresi Gambar Waktu eksekusi: 1.0 detik Ukuran gambar sebelum: 215512 Bytes (210 KB) Ukuran gambar setelah: 191557 Bytes (187 KB) Persentase kompresi: 11.115% Kedalaman pohon: 10 Banyak simpul pada pohon: 311289</p>

Hasil tangkapan layar.





Hasil kompresi gambar.

Metode *Entropy*

Metode *Entropy*

1

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\8.jpg
Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Maximum Pixel Difference (MPD)
4. Entropy

Pilih metode pengukuran error (1-4): 4
Threshold Metode Entropy: (0.5-5)
Masukkan nilai threshold yang di antara batas: 2
Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 10
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\8.jpg
Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\8.jpg

Statistik Kompresi Gambar
Waktu eksekusi: 2.0 detik
Ukuran gambar sebelum: 682502 Bytes (666 KB)
Ukuran gambar setelah: 29281 Bytes (285 KB)
Persentase kompresi: 57.094%
Kedalaman pohon: 11
Banyak simpul pada pohon: 1112445

Hasil tangkapan layar.





Hasil kompresi gambar.

2

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\input\9.png
Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Maximum Pixel Difference (MPD)
4. Entropy

Pilih metode pengukuran error (1-4): 4
Threshold Metode Entropy: (0.5-5)
Masukkan nilai threshold yang di antara batas: 1.5
Masukkan ukuran blok minimum dalam integer (Contoh: 800 untuk blok 40x20): 20
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\9.png
Gambar hasil kompresi berhasil disimpan di: C:\Users\User\OneDrive\Documents\SEM 4\STIMA\Tucil2_13523014\test\output\9.png

Statistik Kompresi Gambar
Waktu eksekusi: 0.0 detik
Ukuran gambar sebelum: 550874 Bytes (537 KB)
Ukuran gambar setelah: 37744 Bytes (36 KB)
Persentase kompresi: 93.148%
Kedalaman pohon: 9
Banyak simpul pada pohon: 67697
```

Hasil tangkapan layar.



Hasil kompresi gambar.

Analisis Algoritma

Kompleksitas Waktu

Pada implementasi algoritma *divide and conquer* dalam persoalan ini, kompleksitas waktu dinilai dari waktu untuk melakukan rekursi, pengecekan setiap node untuk menghitung nilai variansi atau *error*-nya, dan waktu untuk *combine* seluruh upa-persoalan. Kompleksitas waktunya adalah sebagai berikut:

$$T(n) = 4T(n/4) + f(n) + g(m)$$

- $T(n)$ = kompleksitas waktu penyelesaian *image compression* dengan jumlah n pixels pada gambar
- $f(n)$ = kompleksitas waktu perhitungan nilai variansi atau *error* untuk setiap node
- $g(m)$ = kompleksitas waktu untuk COMBINE setiap **leaf** (blok tanpa children/subblok)

Untuk tahap divide, pembagian dapat berlangsung secara instan dengan kompleksitas $O(1)$ sehingga dapat diabaikan. Untuk tahap conquer, rekursi untuk 4 anak pada Quadtree, maka kompleksitasnya $4T(n/4)$, ditambah dengan waktu untuk menghitung *error* pada setiap pixel untuk menentukan apakah blok dibagi atau tidak. Untuk tahap combine dengan menggabungkan gambar, kompleksitasnya $g(m)$ tergantung dari jumlah leaf yang terbentuk pada program.

Pada perhitungan $f(n)$, dipertimbangkan kompleksitas berbagai jenis metode yang dapat digunakan untuk mengukur *error*. Berdasarkan metode variance, MAD, MPD, dan Entropy, hasil kompleksitas algoritma perhitungan *error* adalah $O(n)$ dengan n adalah jumlah pixel yang ada pada blok/node yang diperhitungkan.

Pada perhitungan $g(m)$, jumlah leaf pada kasus terburuk yang terbentuk adalah ketika seluruh leaf terpecah menjadi blok piksel 1×1 terkecil. Dengan demikian, kompleksitasnya mencapai $O(n)$. Sedangkan, kasus terbaik adalah ketika seluruh gambar homogen dan memiliki 1 leaf/blok saja sehingga kompleksitas algoritmanya $O(1)$. Perlu diingat bahwa hubungan jumlah m terhadap jumlah pixel n adalah $m \leq n$.

Dengan asumsi $f(n)$ dan $g(m)$ adalah $O(n)$, menurut teorema master:

$$T(n) = aT(n/b) + cn^d$$

$$O(n^d \log n) \text{ jika } a = b^d$$

Berdasarkan persamaan kompleksitas algoritma di sini, $a = 4$, $b = 4$, dan $d = 1$ (karena c_n berupa $O(n)$). Dengan demikian, dapat diaproksimasi bahwa algoritma *divide and conquer* untuk kompresi gambar dengan metode quadtree memiliki kompleksitas algoritma

$$O(n \log n)$$

Lampiran

Pranala repository: andhikalucas/Tucil2_13523014

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	✓	