LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

Disusun oleh:

Anella Utari Gunadi 13523078

Mayla Yaffa Ludmilla 13523050

SEKOLAH TINGGI ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

FEBRUARI 2025

DAFTAR ISI

1. Deskripsi Masalah

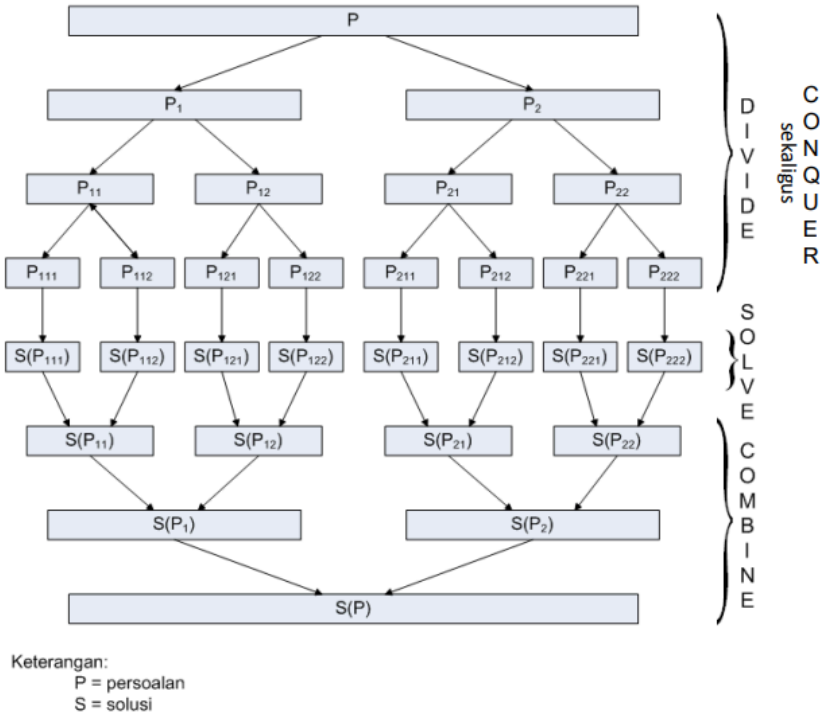
Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya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

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 2 mata kuliah Strategi Algoritma, kami diminta untuk mengembangkan sebuah program kompresi gambar dengan menggunakan struktur data *quadtree*. Implementasi *quadtree* ini didasarkan pada pendekatan algoritma *divide and conquer*. Dalam proses kompresi, pengguna diberikan pilihan untuk menggunakan salah satu dari empat metode perhitungan nilai error, yaitu *Variance, Mean Absolute Deviation (MAD), Max Pixel Difference,* dan *Entropy***.** Masing-masing metode digunakan untuk menentukan apakah suatu blok gambar perlu dibagi lebih lanjut atau tidak dalam konstruksi *quadtree*.

2. Algoritma Divide and Conquer

Algoritma Divide and Conquer terbagi menjadi 3 tahapan yaitu *divide*, *conquer*, dan *combine*. Pada tahap *divide*, persoalan dibagi menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula tetapi berukuran lebih kecil. Idealnya, setiap upa-persoalan berukuran hampir sama. Selanjutnya, di tahap *conquer*, masing-masing upa-persoalan di-*solve* atau diselesaikan secara langsung, jika sudah berukuran kecil, atau secara rekursif jika masih berukuran besar. Terakhir, di tahap *combine*, solusi masing-masing upa-persoalan digabungkan sehingga membentuk solusi persoalan semula.



Di implementasi program kompresi gambar, digunakan algoritma *quadtree*. Proses divide dari algoritma ini adalah dengan membagi gambar menjadi empat blok/kuadran yaitu kuadran kiri atas, kuadran kanan atas, kuadran kiri bawah, dan kuadran kanan bawah secara rekursif hingga mencapai kondisi tertentu. Setiap blok direpresentasikan oleh sebuah simpul atau *node* di dalam *quadtree*, dan ketika blok dibagi menjadi empat, keempat blok turunannya menjadi anak atau *children* dari simpul blok awal.

Proses *conquer* dari algoritma ini adalah mengevaluasi setiap daerah atau blok gambar menggunakan perhitungan error dengan berbagai metode (*variance*, *mean absolute deviation, max pixel difference,* dan *entropy*). Perhitungan error ini menentukan seberapa berbeda warna warna yang ada pada blok gambar tersebut. Jika error di blok gambar lebih kecil dari *threshold* (ambang batas) dan besar blok lebih kecil dari *minimum block size*, warna yang ada pada blok gambar tersebut dianggap sudah cukup homogen. Lalu, warna rata rata dari blok dihitung untuk dijadikan sebagai representasi warna untuk blok tersebut dan simpul yang merepresentasikan blok tersebut menjadi simpul daun.

```
if (error < threshold) or (w <= minBlockSize) or (h <= minBlockSize) then
        { membuat simpul daun dengan warna rata-rata (true bahwa node
adalah daun) }
        node= QuadtreeNode( avg_color(img, x, y, w, h), true )
    else
        halfWidth <- w / 2
        halfHeight <- h / 2
```

```
        { Rekursif dan bagi wilayah menjadi empat (conquer) }
        topLeft     <- quadtree_image(img, x, y, halfWidth, halfHeight,
currentDepth+1, minBlockSize, threshold, method)

        topRight    <-  quadtree_image(img, x + halfWidth, y,  halfWidth,
halfHeight,  currentDepth+1, minBlockSize, threshold, method)

        bottomLeft  <- quadtree_image(img, x, y + halfHeight, halfWidth,
halfHeight,  currentDepth+1, minBlockSize, threshold, method)

        bottomRight <- quadtree_image(img, x + halfWidth, y + halfHeight,
halfWidth, halfHeight, currentDepth+1, minBlockSize, threshold, method)

        { Buat simpul untuk menampung keempat anak}
        node = QuadtreeNode(null, false)
        node.children[0] <- topLeft
        node.children[1] <-topRight
        node.children[2] <- bottomLeft
        node.children[3] <- bottomRight
```

Setelah struktur *quadtree* terbentuk, proses *combine* dari algoritma ini dilakukan dengan menggabungkan simpul-simpul yang ada dalam struktur *quadtree* menjadi sebuah gambar yang utuh. Quadtree ditelusuri, lalu jika menemukan simpul daun, fungsi akan mengisi seluruh area blok pada gambar keluaran dengan warna rata-rata yang sudah dihitung. Jika simpul bukan merupakan simpul daun, fungsi akan menggabungkan hasil dari keempat kuadran anaknya untuk membentuk kembali bagian gambar yang utuh.

```
procedure output_image(node:QuadtreeNode, outputImg:Image, x:integer,
y:integer,
w:integer, h:integer):
{ membuat gambar dengan menggabung simpul-simpul dari quadtree (conquer)
}

Algoritma:
    if ( node adalah daun ) then
        { Isi blok  dengan warna rata-rata yang sudah dihitung }
        fillBlock(outputImg, x, y, w, h, node.color)
    else { cari node daun lainnya }
        halfWidth <- w / 2
        halfHeight <- h / 2

       { lanjut menyusuri quadtree}
        output_image(node.children[0], outputImg, x, y, halfWidth,
halfHeight)
        output_image(node.children[1], outputImg, x + halfWidth, y, w -
halfWidth, halfHeight)
```

```
        output_image(node.children[2], outputImg, x, y + halfHeight,
halfWidth, h - halfHeight)
        output_image(node.children[3], outputImg, x + halfWidth, y +
halfHeight,
                     w - halfWidth, h - halfHeight)
```

3. Implementasi Program (Source Code)

QuadtreeNode.java

```
public class QuadtreeNode {
    int[] color;               // {R,G,B}
    boolean isLeaf;
    QuadtreeNode[] children;

    public QuadtreeNode(int[] color, boolean isLeaf) {
        this.color   = color;
        this.isLeaf  = isLeaf;
        this.children = new QuadtreeNode[4];
    }
}
```

Quadtree.java

```
import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.imageio.stream.ImageOutputStream;
import javax.imageio.stream.FileImageOutputStream;


public class Quadtree {
    private int[][][] img;
    private QuadtreeNode root;
    private int nodeCount = 0;
    private int depth     = 0;

    public Quadtree(int[][][] img, int x, int y, int w, int h, int count,
int minBlockSize,  double threshold, int method) {

        this.img = img;
        this.nodeCount = 0;
        this.depth = 0;
```

```
        this.root = recursive(x, y, w, h, 0, minBlockSize, threshold,
method);
    }

    private QuadtreeNode recursive(int x, int y, int w, int h, int
currentDepth, int minBlockSize, double threshold, int method) {
        depth = Math.max(depth, currentDepth);
        double error;

        if (method == 1) {            // variance
            error = Variance.calculate(img, x, y, w, h);
        } else if (method == 2) {     // max-pixel-diff
            error = MeanAbsoluteDeviation.calculate(img, x, y, w, h);
        } else if (method == 3) {     // max-pixel-diff
            error = MaxPixelDifference.calculate(img, x, y, w, h);
        } else if (method == 4) {     // max-pixel-diff
            error = Entropy.calculate(img, x, y, w, h);
        } else {
            throw new IllegalArgumentException("Unknown method: " +
method);
        }
        if (error < threshold || w <= minBlockSize || h <= minBlockSize)
{
            nodeCount++;
            // BufferedImage frame = new BufferedImage(w,h,
BufferedImage.TYPE_INT_RGB);
            // gifFrames.add(frame);
            return new QuadtreeNode(avgColor(x, y, w, h), true);
        }

        int halfWidth = w / 2, halfHeight = h / 2;
        int[] avg = avgColor(x, y, w, h);
        QuadtreeNode node = new QuadtreeNode(avg, false);
        node.children[0] = recursive(x, y, halfWidth, halfHeight,
currentDepth+1, minBlockSize, threshold, method);
        node.children[1] = recursive(x + halfWidth, y,  halfWidth,
halfHeight, currentDepth+1, minBlockSize, threshold, method);
        node.children[2] = recursive(x, y + halfHeight, halfWidth,
halfHeight, currentDepth+1, minBlockSize, threshold, method);
        node.children[3] = recursive(x + halfWidth, y + halfHeight,
halfWidth, halfHeight, currentDepth+1, minBlockSize, threshold, method);
        nodeCount++;

        return node;
    }


    private int[] avgColor(int x, int y, int width, int height) {
        int r=0,g=0,b=0;
        for (int i=y;i<y+height;i++)
            for (int j=x;j<x+width;j++) {
                r += img[i][j][0];
                g += img[i][j][1];
```

```
                b += img[i][j][2];
            }
        int totalPixel = width*height;
        return new int[]{ r/totalPixel, g/totalPixel, b/totalPixel };
    }

    public void saveImage(int w, int h, String pathImage, String pathGif)
{
        try {
            BufferedImage outputImage = new BufferedImage(w, h,
BufferedImage.TYPE_INT_RGB);
            createOutputImage(root, outputImage, 0, 0, w, h);
            ImageIO.write(outputImage, "jpg", new File(pathImage));

            List<BufferedImage> gifFrames = new ArrayList<>();
            for (int d = 0; d <= this.depth; d++) {
                BufferedImage frame = new BufferedImage(w, h,
BufferedImage.TYPE_INT_RGB);
                renderUpToDepth(root, frame, 0, 0, w, h, 0, d);
                gifFrames.add(frame);
            }

            createGif(gifFrames, pathGif);
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void saveCompressedImage(int w, int h, String path) {
        try {
            BufferedImage out = new BufferedImage(w, h,
BufferedImage.TYPE_INT_RGB);
            createOutputImage(root, out, 0, 0, w, h);
            ImageIO.write(out, "jpg", new File(path));
        } catch (IOException ex) {
            ex.printStackTrace();
        }
    }

    public void createGif(List<BufferedImage> gifFrames, String
outputPath ) throws IOException{
        ImageOutputStream output = new FileImageOutputStream(new
File(outputPath));
        GifWriter gifWriter = new GifWriter(output,
gifFrames.get(0).getType());
        for (BufferedImage frame : gifFrames) {
            gifWriter.writeToSequence(frame);
        }
        gifWriter.close();
        output.close();
    }

    private void renderUpToDepth(QuadtreeNode node, BufferedImage out,
```

```
int x, int y, int width, int height,
                                int currentDepth, int maxDepthAllowed) {
        // jika node yg ditemukan adalah leaf node atau di depth yang >=
maxdepthAllowed, ambil warna untuk dipakai di node itu
        // jika tidak, rekursi ke anak-anak dari node
        if (node.isLeaf || currentDepth >= maxDepthAllowed) {
            Color c = new Color(node.color[0], node.color[1],
node.color[2]);
            for (int i=0; i<height; i++) {
                for (int j=0; j<width; j++) {
                    out.setRGB(x+j, y+i, c.getRGB());
                }
            }
        } else {
            int halfWidth = width / 2;
            int halfHeight = height / 2;
            int remWidth = width - halfWidth;
            int remHeight = height - halfHeight;

            renderUpToDepth(node.children[0], out, x, y, halfWidth,
halfHeight, currentDepth+1, maxDepthAllowed);
            renderUpToDepth(node.children[1], out, x+halfWidth, y,
remWidth, halfHeight, currentDepth+1, maxDepthAllowed);
            renderUpToDepth(node.children[2], out, x, y+halfHeight,
halfWidth, remHeight, currentDepth+1, maxDepthAllowed);
            renderUpToDepth(node.children[3], out, x+halfWidth,
y+halfHeight, remWidth, remHeight, currentDepth+1, maxDepthAllowed);
        }

}


    private void createOutputImage(QuadtreeNode n, BufferedImage
outputImage, int x, int y, int width, int height) {
        if (n.isLeaf) {
            Color c = new Color(n.color[0], n.color[1], n.color[2]);
            for (int i=0;i<height;i++) {
                for (int j=0;j<width;j++){
                    if (y+i < outputImage.getHeight() && x+j <
outputImage.getWidth()){
                        outputImage.setRGB(x+j, y+i, c.getRGB());
                    }
                }
            }

        } else {
            int halfWidth = width/2, halfHeight = height/2;
            int remWidth = width-halfWidth, remHeight =
height-halfHeight;
            createOutputImage(n.children[0], outputImage, x, y,
halfWidth, halfHeight);
            createOutputImage(n.children[1], outputImage, x+halfWidth, y,
remWidth, halfHeight);
```

```
            createOutputImage(n.children[2], outputImage, x,
y+halfHeight, halfWidth, remHeight);
            createOutputImage(n.children[3], outputImage, x+halfWidth,
y+halfHeight, remWidth, remHeight);
        }
    }

    public int getDepth(){
        return depth;
    }
    public int getNodeCount() {
        return nodeCount;
    }
}
```

Variance.java

```
public class Variance {
    public static double calculate(int[][][] img,
                                   int x, int y, int w, int h) {

        int[] avg = avgColor(img, x, y, w, h);
        int avgr = avg[0];
        int avgg = avg[1];
        int avgb = avg[2];

        double sr=0, sg=0, sb=0;
        for (int i=y;i<y+h;i++)
            for (int j=x;j<x+w;j++) {
                sr += Math.pow(img[i][j][0] - avgr, 2);
                sg += Math.pow(img[i][j][1] - avgg, 2);
                sb += Math.pow(img[i][j][2] - avgb, 2);
            }

        double n = w * h;
        double vr = sr/n, vg = sg/n, vb = sb/n;
        return (vr + vg + vb) / 3.0;
    }
    private static int[] avgColor(int[][][] img,
                                  int x, int y, int w, int h) {
        int r=0,g=0,b=0;
        for (int i=y;i<y+h;i++)
            for (int j=x;j<x+w;j++) {
                r += img[i][j][0];
                g += img[i][j][1];
                b += img[i][j][2];
            }
        int total = w*h;
        return new int[]{ r/total, g/total, b/total };
    }
```

```
}
```

MeanAbsoluteDeviation.java

```
public class MeanAbsoluteDeviation {
    public static double calculate(int[][][] rgbImage, int x, int y, int
width, int height) {
        return (calculateMAD(rgbImage, x, y, width, height, 0) +  // Red
                calculateMAD(rgbImage, x, y, width, height, 1) +  //
Green
                calculateMAD(rgbImage, x, y, width, height, 2))   // Blue
                / 3.0;
    }

    private static double calculateMAD(int[][][] rgbImage, int x, int y,
int width, int height, int channel) {
        int sum = 0, count = 0;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                sum += rgbImage[i][j][channel];
                count++;
            }
        }

        double mean = sum / (double) count;

        double mad = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                mad += Math.abs(rgbImage[i][j][channel] - mean);
            }
        }

        return mad / count;
    }
}
```

MaxPixelDifference.java

```
public class MaxPixelDifference {

    public static double calculate(int[][][] img, int x, int y, int w,
int h) {

        int dr = range(img, x, y, w, h, 0);
        int dg = range(img, x, y, w, h, 1);
        int db = range(img, x, y, w, h, 2);
```

```
        return (dr + dg + db) / 3.0;
    }

    private static int range(int[][][] img, int x, int y, int w, int h,
int ch) {
        int max = Integer.MIN_VALUE;
        int min = Integer.MAX_VALUE;
        for (int i=y;i<y+h;i++)
            for (int j=x;j<x+w;j++) {
                int v = img[i][j][ch];
                if (v > max) max = v;
                if (v < min) min = v;
            }
        return max - min;
    }
}
```

Entropy.java

```
import java.util.HashMap;
import java.util.Map;

public class Entropy {
    public static double calculate(int[][][] rgbImage, int x, int y, int
width, int height) {
        return (calculateEntropy(rgbImage, x, y, width, height, 0) +  //
Red
                calculateEntropy(rgbImage, x, y, width, height, 1) +  //
Green
                calculateEntropy(rgbImage, x, y, width, height, 2))  //
Blue
                / 3.0;
    }

    private static double calculateEntropy(int[][][] rgbImage, int x, int
y, int width, int height, int channel) {
        Map<Integer, Integer> histogram = new HashMap<>();
        int totalPixels = width * height;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                int value = rgbImage[i][j][channel];
                histogram.put(value, histogram.getOrDefault(value, 0) +
1);
            }
        }

        double entropy = 0;
        for (int count : histogram.values()) {
            double probability = count / (double) totalPixels;
```

```
            entropy -= probability * (Math.log(probability) /
Math.log(2));
        }
        return entropy;
    }
}
```

CompressionResult.java

```
import java.io.File;

public class CompressionResult {
    public double threshold;
    public int minBlockSize;

    public CompressionResult(double threshold, int minBlockSize) {
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
    }

    public static CompressionResult CompressionTarget(int[][][]
imageArray, int width, int height, double targetCompression, int
errorMethod, long originalSize) {
        double bestThreshold = 0;
        int bestBlockSize = 1;
        double bestCompression = 0;
        long bestSize = Long.MAX_VALUE;

        double maxThreshold = getMaxThreshold(errorMethod);
        double tolerance = 0.001;

        for (int blockSize : new int[]{1, 4, 8, 16, 32}) {
            double low = 0;
            double high = maxThreshold;

            while (low <= high) {
                double mid = (low + high) / 2.0;

                Quadtree qt = new Quadtree(imageArray, 0, 0, width,
height, 0, blockSize, mid, errorMethod);
                qt.saveCompressedImage(width, height,
"temp_compress.jpg");

                File temp = new File("temp_compress.jpg");
                long size = temp.length();
                double compression = 1 - ((double) size / originalSize);

                if (compression >= targetCompression) {
                    if (Math.abs(compression - targetCompression) <
Math.abs(bestCompression - targetCompression)) {
```

```
                        bestThreshold = mid;
                        bestBlockSize = blockSize;
                        bestCompression = compression;
                        bestSize = size;
                    }
                    high = mid - tolerance;
                } else {
                    low = mid + tolerance;
                }
                temp.delete();
            }
        }
        return new CompressionResult(bestThreshold, bestBlockSize);
    }

    public static double getMaxThreshold(int method) {
        return switch (method) {
            case 1 -> 65025.0;
            case 2, 3 -> 255.0;
            case 4 -> 8.0;
            default -> 255.0;
        };
    }
}
```

GifWriter.java

```
// package com.memorynotfound.image;

import javax.imageio.*;
import javax.imageio.metadata.IIOInvalidTreeException;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.stream.ImageOutputStream;
import java.awt.image.RenderedImage;
import java.io.IOException;

public class GifWriter {

    protected ImageWriter imgWriter;
    protected ImageWriteParam imgParams;
    protected IIOMetadata metadata;

    public GifWriter(ImageOutputStream out, int imageType) throws
IOException {
        imgWriter = ImageIO.getImageWritersBySuffix("gif").next();
        imgParams = imgWriter.getDefaultWriteParam();

        ImageTypeSpecifier imageTypeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
        metadata = imgWriter.getDefaultImageMetadata(imageTypeSpecifier,
imgParams);
```

```
        configureRootMetadata();

        imgWriter.setOutput(out);
        imgWriter.prepareWriteSequence(null);
    }

    private void configureRootMetadata() throws IIOInvalidTreeException {
        String metaFormatName = metadata.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode)
metadata.getAsTree(metaFormatName);

        IIOMetadataNode graphicsControlExtensionNode = getNode(root,
"GraphicControlExtension");
        graphicsControlExtensionNode.setAttribute("disposalMethod",
"none");
        graphicsControlExtensionNode.setAttribute("userInputFlag",
"FALSE");
        graphicsControlExtensionNode.setAttribute("transparentColorFlag",
"FALSE");
        graphicsControlExtensionNode.setAttribute("delayTime",
Integer.toString(1000 / 10));
        graphicsControlExtensionNode.setAttribute("transparentColorIndex",
"0");

        IIOMetadataNode appExtensionsNode = getNode(root,
"ApplicationExtensions");
        IIOMetadataNode child = new
IIOMetadataNode("ApplicationExtension");
        child.setAttribute("applicationID", "NETSCAPE");
        child.setAttribute("authenticationCode", "2.0");

        child.setUserObject(new byte[]{ 0x1, (byte) (0xFF), (byte)
(0xFF)});
        appExtensionsNode.appendChild(child);
        metadata.setFromTree(metaFormatName, root);
    }

    private static IIOMetadataNode getNode(IIOMetadataNode rootNode, String
nodeName){
        int nNodes = rootNode.getLength();
        for (int i = 0; i < nNodes; i++){
            if (rootNode.item(i).getNodeName().equalsIgnoreCase(nodeName)){
                return (IIOMetadataNode) rootNode.item(i);
            }
        }
        IIOMetadataNode node = new IIOMetadataNode(nodeName);
        rootNode.appendChild(node);
        return(node);
    }

    public void writeToSequence(RenderedImage img) throws IOException {
        imgWriter.writeToSequence(new IIOImage(img, null, metadata),
imgParams);
    }
```

```
    public void close() throws IOException {
        imgWriter.endWriteSequence();
    }

}
```

4. Hasil Implementasi Program

4.1. Tangkapan Layar Input dan Output

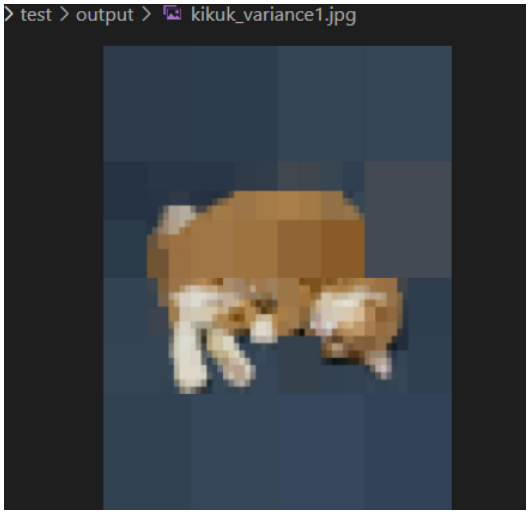
4.1.1. Input dan Output Metode Variance

- Input dan Output

```
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Mayla\Documents\Kuliah\Se
mester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 1
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan target): 0
Masukkan ambang batas (0 - 65025): 500
Masukkan ukuran minimum blok: 20
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Mayla\Documents\Kuliah\Semester-
4\Strategi-Algoritma\Tucil2_13523050_13523078\test\output\kikuk_variance1.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 2

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 270 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 53615 bytes
Persentase kompresi: 66.84947938565034%
Kedalaman pohon: 6
Banyak simpul pada pohon: 677
```

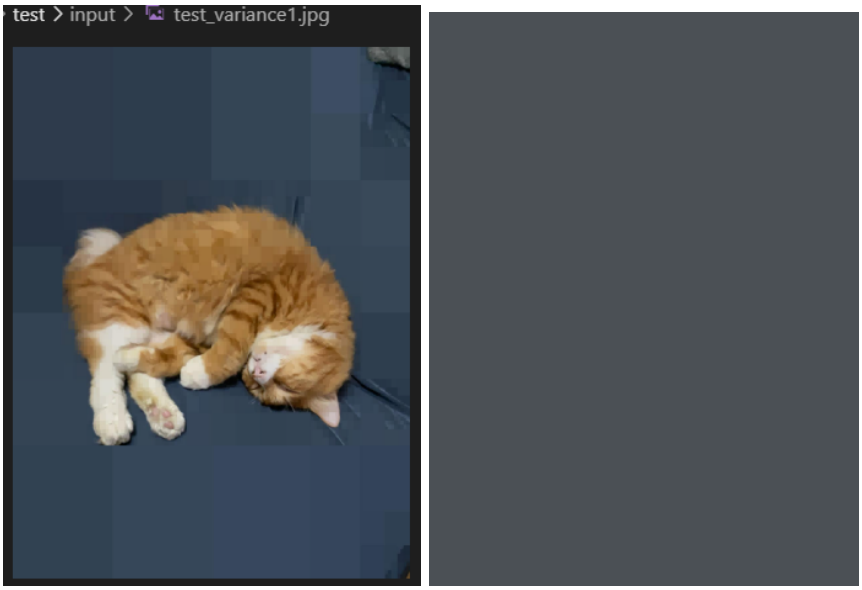


- Input dan Output 2

```
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Mayla\Documents\Kuliah\Se
mester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 1
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan target): 0.5
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Mayla\Documents\Kuliah\Semester-
078\test\input\test_variance1.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 1
Masukkan path absolut untuk output GIF animasi: C:\Users\Mayla\Documents\Kuliah\Semester
-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\output\variance.gif

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 21868 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 80865 bytes
Persentase kompresi: 50.000618306828585%
Kedalaman pohon: 10
Banyak simpul pada pohon: 40177
```



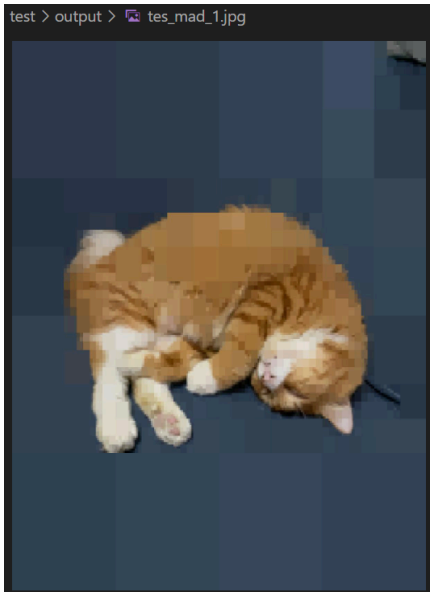
4.1.2. Input dan Output Metode Mean Absolute Deviation

- Input dan Output 1

```
C:\Users\Anella Utari\Documents\GitHub\Tucil2_13523050_13523078>java -cp bin ImagePr
ocessor
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Anella Utari\Document
s\GitHub\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 2
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan target): 0
Masukkan ambang batas (0 - 255): 10
Masukkan ukuran minimum blok: 4
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Anella Utari\Documents\GitHu
b\Tucil2_13523050_13523078\test\output\tes_mad_1.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 2

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 151 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 72424 bytes
Persentase kompresi: 55.219746246877556%
Kedalaman pohon: 8
Banyak simpul pada pohon: 6793
```

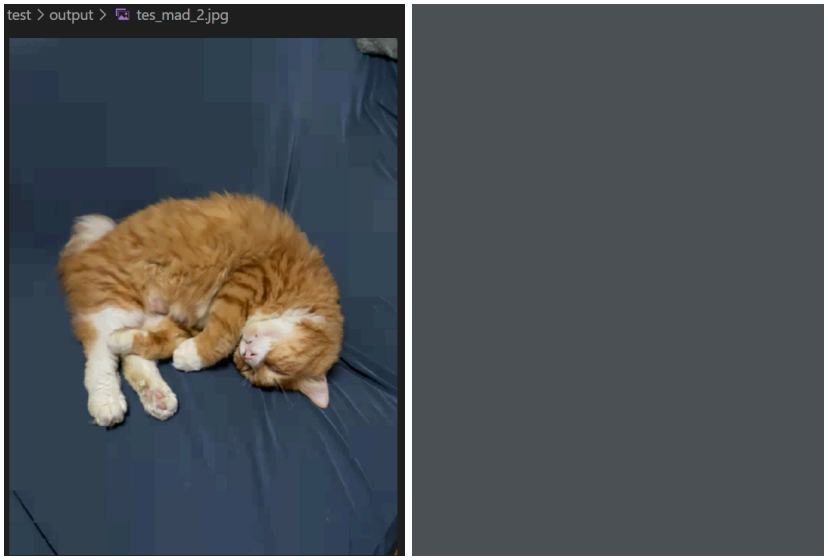


- Input dan Output 2

```
C:\Users\Anella Utari\Documents\GitHub\Tucil2_13523050_13523078>java -cp bin ImageProc
essor
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Anella Utari\Documents\
GitHub\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 2
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan target): 0.43
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Anella Utari\Documents\GitHub\
Tucil2_13523050_13523078\test\output\tes_mad_2.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 1
Masukkan path absolut untuk output GIF animasi: C:\Users\Anella Utari\Documents\GitHub
\Tucil2_13523050_13523078\test\output\tes_mad_2.gif

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 16124 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 92163 bytes
Persentase kompresi: 43.01498775752479%
Kedalaman pohon: 10
Banyak simpul pada pohon: 161077
```



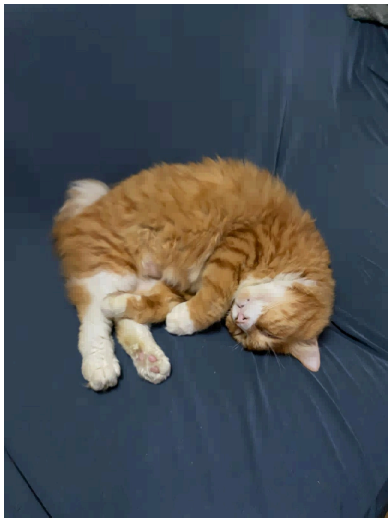
4.1.3. Input dan Output Metode Max Pixel Differentiation

- Input dan Output 1

```
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Mayla\Documen
ts\Kuliah\Semester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\input\
kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 3
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan targ
et): 0
Masukkan ambang batas (0 - 255): 20
Masukkan ukuran minimum blok: 1
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Mayla\Documents\Kuli
ah\Semester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\output\test-m
pd.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 2

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 283 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 95146 bytes
Persentase kompresi: 41.17057848786882%
Kedalaman pohon: 10
Banyak simpul pada pohon: 71529
```

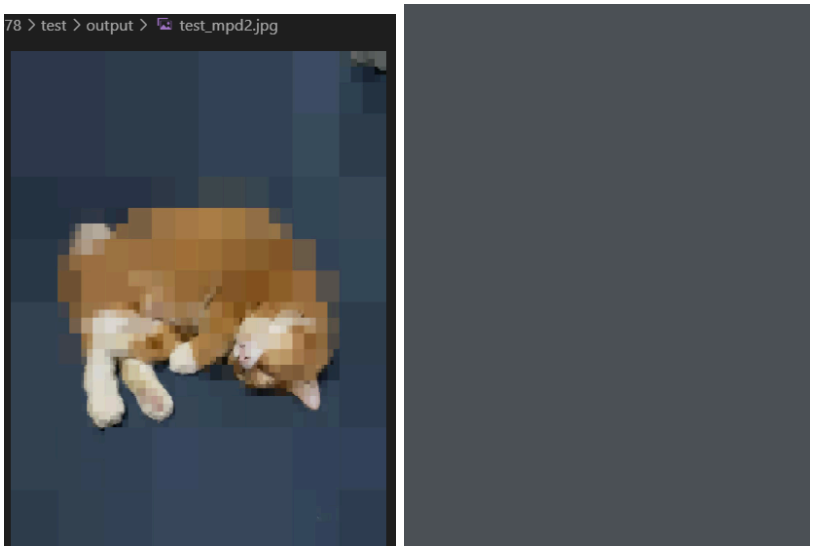


- Input dan Output 2

```
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Mayla\Documen
ts\Kuliah\Semester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\input\
kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 3
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan targ
et): 0.63
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Mayla\Documents\Kuli
ah\Semester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\output\test_m
pd2.jpg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 1
Masukkan path absolut untuk output GIF animasi: C:\Users\Mayla\Documents\Kul
iah\Semester-4\Strategi-Algoritma\Tucil2_13523050_13523078\test\output\mpd.g
if

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 16563 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 59814 bytes
Persentase kompresi: 63.0165953552791%
Kedalaman pohon: 10
Banyak simpul pada pohon: 2093
```



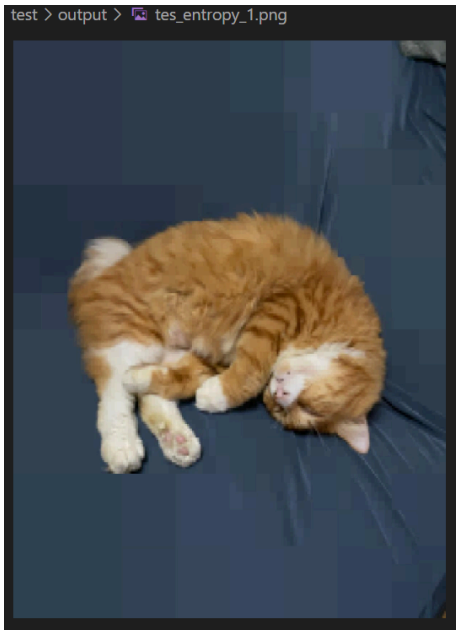
4.1.4 Input dan Output Metode Entropy

- Input dan Output 1

```
C:\Users\Anella Utari\Documents\GitHub\Tucil2_13523050_13523078>java -cp bin ImageProc
essor
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Anella Utari\Documents\
GitHub\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 4
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan target): 0
Masukkan ambang batas (0 - 8): 4
Masukkan ukuran minimum blok: 8
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Anella Utari\Documents\GitHub\
Tucil2_13523050_13523078\test\output\tes_entropy_1.png
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 2

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 519 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 88979 bytes
Persentase kompresi: 44.98367669972547%
Kedalaman pohon: 8
Banyak simpul pada pohon: 21193
```

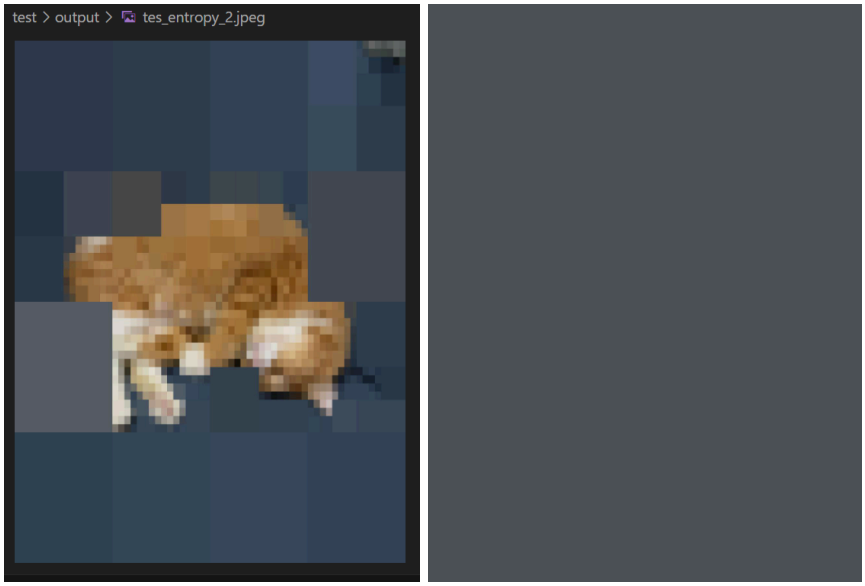


- Input dan Output 2

```
C:\Users\Anella Utari\Documents\GitHub\Tucil2_13523050_13523078>java -cp bi
n ImageProcessor
Masukkan alamat absolut gambar yang ingin dikompresi: C:\Users\Anella Utari
\Documents\GitHub\Tucil2_13523050_13523078\test\input\kikuk.jpg
Metode perhitungan error:
1. Variance
2. Mean Absolute Deviaton (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan metode perhitungan error (1-4): 4
Masukkan target kompresi (dalam desimal antara 0-1, 0 untuk nonaktifkan tar
get): 0.64
Masukkan alamat absolut gambar hasil kompresi: C:\Users\Anella Utari\Docume
nts\GitHub\Tucil2_13523050_13523078\test\output\tes_entropy_2.jpeg
Apakah Anda ingin menyimpan output dalam bentuk GIF animasi?
1. Ya
2. Tidak
Masukkan pilihan Anda (1/2): 1
Masukkan path absolut untuk output GIF animasi: C:\Users\Anella Utari\Docum
ents\GitHub\Tucil2_13523050_13523078\test\output\tes_entropy_2.gif

------------------------------------------------------------------------
                    Mohon tunggu. Sedang diproses...
------------------------------------------------------------------------

Waktu eksekusi: 50456 ms
Ukuran gambar sebelum kompresi: 161732 bytes
Ukuran gambar setelah kompresi: 58153 bytes
Persentase kompresi: 64.0436029975515%
Kedalaman pohon: 6
Banyak simpul pada pohon: 1053
```



4.2. Hasil Analisis Percobaan

Dalam algoritma *divide and conquer* yang diimplementasikan pada pembuatan struktur Quadtree, algoritma ini memiliki kompleksitas waktu yang bergantung pada jumlah pembagian yang terjadi, yang dipengaruhi oleh ukuran gambar (n x n) dan parameter seperti *threshold* dan *minBlockSize*. Pada kasus terburuk, yaitu jika *threshold* sangat kecil dan *minBlockSize* adalah 1, setiap blok akan terus dibagi hingga ukuran minimum yang menghasilkan pembagian penuh menjadi piksel-piksel individual. Dalam hal ini, kompleksitas waktu menjadi $O(n^2)$ karena setiap piksel diperiksa dan dihitung *error*-nya. Pada kasus terbaik, yaitu jika *threshold* tinggi, rekursi akan berhenti lebih awal dan blok tidak akan dibagi lagi. Dalam hal ini, total kedalaman pohon akan sedikit dan kompleksitas waktunya mendekati O(1) atau O(log n) tergantung seberapa sering gambar dibagi.

Pada perbandingan metode error yang dipakai, metode *Variance* dan *Entropy* biasanya membagi gambar menjadi lebih banyak bagian karena keduanya memperhitungkan sebaran nilai piksel secara statistik. Oleh karena itu, kedua metode ini lebih peka terhadap perbedaan warna yang kecil yang mengakibatkan saat nilai *threshold*-nya kecil, struktur Quadtree yang dihasilkan akan punya banyak simpul dan kedalaman yang cukup tinggi. Sedangkan, metode MAD dan *Max Pixel Difference* lebih sederhana dan tidak terlalu sensitif terhadap perubahan warna yang kecil. Jadi saat menggunakan *threshold* yang sama, kedua metode ini biasanya menghasilkan pembagian blok yang sedikit lebih rendah dibandingkan *Variance* dan *Entropy*.

Nilai *threshold* dan *minimum block size* juga sangat berpengaruh terhadap hasil kompresi gambar menggunakan Quadtree. Semakin kecil nilai *threshold*, maka akan semakin banyak bagian gambar yang berbeda dan perlu dibagi lagi. Hal ini membuat kedalaman pohon lebih tinggi dan ukuran hasil kompresi semakin besar. Sedangkan, jika nilai *minimal block size* semakin besar, maka pembagian blok akan berhenti lebih cepat yang membuat simpul menjadi terbentuk lebih sedikit dan kompresi menjadi lebih tinggi.

5. Penjelasan Implementasi Bonus

5.1. Implementasi Target Persentase Kompresi

Fitur target persentase kompresi ini adalah fitur yang memungkinkan pengguna untuk menentukan target persentase kompresi secara langsung tanpa harus mengatur nilai *threshold* dan *minimum block size* secara manual. Program akan secara otomatis mencari nilai *threshold* dan *minimum block size* yang paling sesuai agar hasil kompresi mendekati target yang diinginkan.

Program pada awalnya menginisialisasi nilai `threshold`, `minBlockSize`, `compression rate`, ukuran file hasil kompresi terbaik sejauh ini (`bestThreshold`, `bestBlockSize`, `bestCompression`, `bestSize`), dan `tolerance` sebagai nilai (delta) yang mengatur seberapa presisi pencarian *threshold* dan *minimal block size* dilakukan. Program akan menguji beberapa nilai minBlockSize yang umum digunakan, yaitu 1, 4, 8, 16, dan 32. Setiap nilai minBlockSize yang diuji, program akan melakukan pencarian nilai threshold menggunakan metode *binary search* dengan cara mengatur nilai awal low = 0 dan high = maxThreshold yang bergantung pada metode error yang digunakan. Kemudian ambil nilai tengah dengan cara `mid = (low+high) / 2.0` sebagai threshold sementara.

```
for (int blockSize : new int[]{1, 4, 8, 16, 32})
            double low <-  0
            double high <- maxThreshold

            while (low <= high) do
                double mid = (low + high) / 2.0
```

Setelah mendapatkan threshold dan minBlockSize, buat objek Quadtree dengan menggunakan parameter mid sebagai threshold sementara dan nilai minBlockSize yang pada saat itu sedang diuji dalam iterasi. Gambar yang telah dikompresi disimpan sementara untuk kemudian dihitung persentase kompresinya. Jika persentase kompresi sementara masih kurang dari target yang diinginkan oleh pengguna, maka program akan

menaikkan batas bawah dengan menambahkan nilai `mid` oleh `tolerance`. Jika persentase kompresi sementara sudah melebihi target yang diinginkan oleh pengguna, maka program akan menurunkan batas atas dengan mengurangi nilai `mid` oleh `tolerance`. Tetapi, jika persentase kompresi telah mendekati target serta lebih mendekati target daripada hasil sebelumnya, maka program akan menyimpan nilai *threshold* dan *minimal block size*sebagai `bestThreshold` dan `bestBlockSize.` File gambar sementara akan dihapus setiap iterasi. Jika seluruh iterasi telah dilakukan, maka program akan mengembalikan nilai `bestThreshold` dan `bestBlockSize` sebagai *threshold* dan *minimal block size* yang digunakan dalam pemrosesan kompresi gambar.

```
if (compression >= targetCompression) then
                    if (Math.abs(compression - targetCompression) <
Math.abs(bestCompression - targetCompression)) then
                        bestThreshold <- mid
                        bestBlockSize <- blockSize
                        bestCompression <- compression
                        bestSize <- size
                    high <- mid - tolerance
                else
                    low <- mid + tolerance

                temp.delete()
```

5.2 Implementasi Gif Proses Kompresi

Fitur *process GIF* ini berfungsi untuk menampilkan animasi tahapan kompresi gambar menggunakan Quadtree, mulai dari blok besar hingga terbentuk detail penuh. Pada awalnya program menciptakan daftar frame untuk GIF melalui loop yang berjalan dari kedalaman 0 hingga kedalaman maksimum Quadtree yang telah dibangun. Setiap iterasi membuat frame baru yang menunjukkan hasil kompresi sampai kedalaman tertentu, sehingga setiap sekuens frame GIF akan menampilkan gambar yang lebih detail dibandingkan dengan sebelumnya.

```
for (int d = 0; d <= this.depth; d++) {
    BufferedImage frame = new BufferedImage(w, h,
BufferedImage.TYPE_INT_RGB);
    renderUpToDepth(root, frame, 0, 0, w, h, 0, d);
    gifFrames.add(frame);
}
```

Setelah semua *frame* terkumpul, program menggunakan kelas GifWriter yang merupakan modifikasi dari program open source GifSequenceWriter untuk menulis setiap *frame* dalam format GIF secara berurutan dengan pengaturan metadata seperti *delay time* dan *disposal method*. Setelah konstruksi writer, fungsi konstruktor mengonfigurasi

beberapa properti pada metadata GIF seperti *delayTime*, disposal method, dan instruksi untuk looping GIF.

```
    imgWriter = ImageIO.getImageWritersBySuffix("gif").next();
        imgParams = imgWriter.getDefaultWriteParam();

        ImageTypeSpecifier imageTypeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
        metadata = imgWriter.getDefaultImageMetadata(imageTypeSpecifier,
imgParams);

        configureRootMetadata();

        imgWriter.setOutput(out);
        imgWriter.prepareWriteSequence(null);
```

6. Lampiran

- Link repository

  https://github.com/maymilla/Tucil2_13523050_13523078

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	✓	