

LAPORAN TUGAS KECIL 2

KOMPRESI GAMBAR DENGAN METODE QUADTREE

Disusun untuk Memenuhi Tugas Kecil 2 Mata Kuliah Strategi Algoritma IF2211

Dosen Pengampu:

Dr. Ir. Rinaldi Munir, M.T.



Karol Yangqian Poetrachya 13523093

Aria Judhistira 13523112

Kelas K2

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	3
DESKRIPSI MASALAH.....	3
1.1. Deskripsi Masalah.....	3
1.2. Algoritma.....	3
1.3. Pseudocode.....	5
BAB II	
IMPLEMENTASI.....	7
2.1. Implementasi Node.java.....	7
2.2. Implementasi Quadtree.java.....	7
2.3. Implementasi IOHandler.java.....	9
2.4. Implementasi Main.java.....	10
BAB III	
SOURCE CODE.....	12
3.1. Repository Program.....	12
3.2. Source Code Program.....	12
3.2.1. Node.java.....	12
3.2.2. Quadtree.java.....	12
3.2.3. IOHandler.java.....	30
3.2.4. Main.java.....	37
BAB IV	
PERCOBAAN DAN ANALISIS.....	46
4.1. Percobaan.....	46
4.1.1. Test Case 1: Metode Error Variansi.....	46
4.1.2. Test Case 2: Metode Error Mean Absolute Deviation.....	46
4.1.3. Test Case 3: Metode Error Maximum Pixel Difference.....	47
4.1.4. Test Case 4: Metode Error Entropi.....	48
4.1.5. Test Case 5: Gambar dengan Kanal Alpha (Transparansi).....	49
4.1.6. Test Case 6: Target Kompresi (Bonus).....	50
4.1.7. Test Case 7: GIF (Bonus).....	51
4.2. Analisis.....	53
4.2.1. Kompleksitas Algoritma.....	53
4.2.2. Analisis Percobaan.....	53
LAMPIRAN.....	56
REFERENSI.....	57

BAB I

DESKRIPSI MASALAH

1.1. Deskripsi Masalah

Kompresi gambar merupakan suatu proses untuk mengurangi ukuran suatu file citra tanpa mengurangi kualitasnya secara signifikan. Cara untuk mencapai itu dapat berupa mengurangi redundansi data yang terkandung pada file gambar. Salah satu metode untuk melakukan kompresi gambar adalah dengan memanfaatkan struktur data Quadtree. Quadtree sendiri adalah struktur data pohon yang memiliki maksimal empat anak. Dalam konteks ini, Quadtree akan secara rekursif membagi gambar menjadi 4 blok dan mengevaluasi errornya. Jika errornya melebihi ambang batas yang ditentukan, Quadtree akan membagi anak tersebut menjadi 4 sub-blok lagi. Jika errornya berada di bawah atau sama dengan ambang batasnya, atau ukuran blok setelah dibagi 4 akan menjadi kurang dari ukuran blok minimum yang ditentukan, proses dihentikan dan *node* tersebut dijadikan *leaf*. Setiap daun Quadtree akan diisi dengan warna rata-rata piksel-piksel pada daun tersebut.

1.2. Algoritma

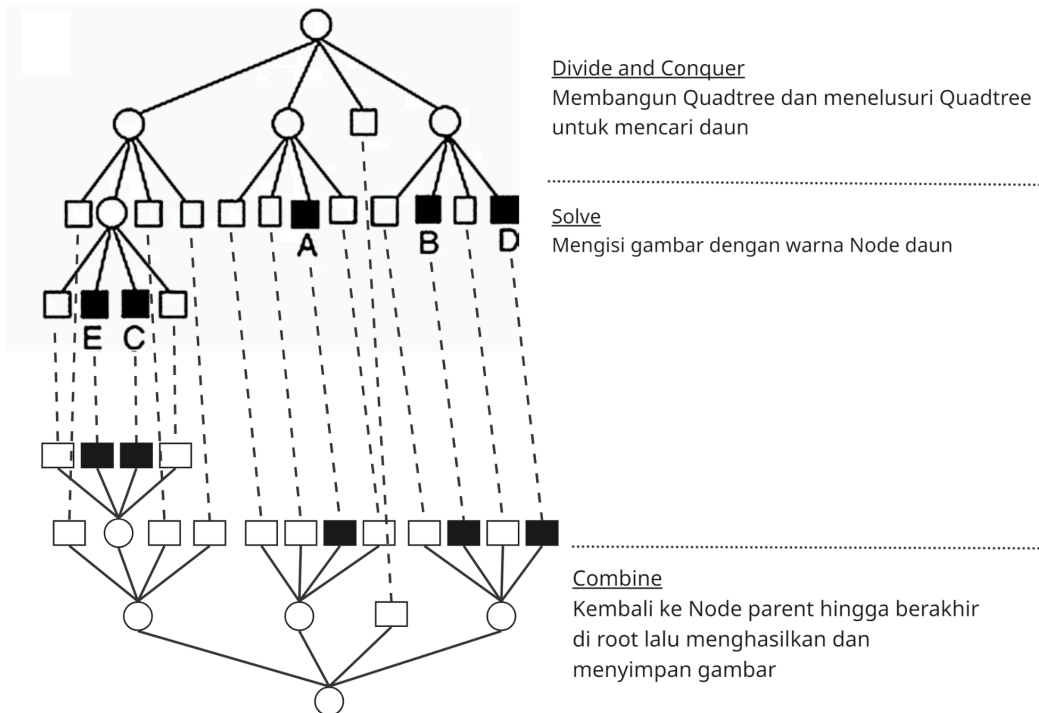
Penyelesaian permasalahan kompresi gambar menggunakan Quadtree menggunakan algoritma *divide and conquer*. Sederhananya, algoritma ini akan membagi suatu persoalan menjadi upa-upa-persoalan yang lebih kecil tetapi mencakup persoalan yang sama. Kemudian, algoritma akan menyelesaikan masing-masing upa-persoalan tersebut secara rekursif dan menggabungkan semua solusi upa-persoalan menjadi satu solusi utuh yang menyelesaikan permasalahan awal.

Berikut adalah langkah-langkah penyelesaian dengan algoritma *divide and conquer* untuk persoalan ini.

1. Program meminta semua *input* dari pengguna yang dibutuhkan, yakni alamat absolut gambar, metode perhitungan error, ambang batas error, ukuran blok terkecil yang diperbolehkan, dan alamat absolut gambar setelah melakukan kompresi.

2. Program akan menginisialisasi objek Quadtree dengan atribut-atributnya yang diatur menjadi nilai *default* dan sebuah objek Node sebagai akarnya. Secara garis besar, kelas Quadtree bertanggungjawab untuk program kompresi keseluruhan dengan membangun Quadtree itu sendiri, sedangkan kelas Node bertanggungjawab menjadi *node* pada Quadtree yang merepresentasikan suatu blok pada gambar.
3. Pembangunan Quadtree dilakukan secara rekursif dimulai dari Node root (root diinisialisasi dengan posisi blok (0, 0), dimensi gambar, warna rata-rata blok, error blok, dan tingkat kedalaman 0). Program akan mengecek apakah Node yang dievaluasi dapat dijadikan daun atau tidak. Node dijadikan daun apabila ukurannya kurang dari atau sama dengan satu, ukurannya ketika dibagi 4 kurang dari ukuran blok terkecil yang diberikan, atau error kurang dari ambang batas. Jika belum memenuhi kriteria menjadi daun, maka Node tersebut dijadikan Node yang “beranak” 4 Node. Setiap Node anak diberikan posisi, dimensi blok pada gambar, warna rata-rata blok, error blok, dan tingkat kedalamannya.
4. Setelah memberikan setiap Node anak nilai-nilai untuk atribut mereka, program akan melanjutkan pembangunan Quadtree secara rekursif melalui Node-Node anak tersebut. Proses ini dilakukan hingga ditemukan daun Quadtree terakhir.
5. Setelah pembangunan struktur data Quadtree selesai, setiap daun Quadtree akan digabungkan menjadi satu gambar yang utuh.
6. Proses pengisian data warna pada Quadtree dilakukan secara rekursif juga. Program akan mengecek apakah suatu Node merupakan daun atau tidak. Jika bukan, program akan melanjutkan proses kepada Node-Node anaknya. Jika merupakan daun, program akan mengisi gambar pada posisi dengan dimensi Node daun tersebut dengan warna yang dimilikinya (yakni warna rata-rata blok daun tersebut).
7. Proses pengisian data warna dilanjutkan hingga mencapai daun terakhir. Pada tahap ini, setiap blok gambar sudah diberi warna yang sesuai dengan

Node daun yang merepresentasikannya dan gambar sudah terbentuk secara utuh.



1.3. Pseudocode

Terdapat dua prosedur utama dalam penyelesaian persoalan ini, yakni *BuildQuadtree* dan *FillImageData*. Prosedur *BuildQuadtree* bertujuan untuk membangun Quadtree berdasarkan gambar secara rekursif. Prosedur *FillImageData* berfungsi untuk membuat gambar (*BufferedImage*) baru dari Quadtree yang telah dibangun. Dalam konteks *divide and conquer*, prosedur *BuildQuadtree* berperan pada bagian *divide* dan *conquer*, sedangkan prosedur *FillImageData* berperan untuk menggabungkan upa-upa solusi berupa daun-daun pada Quadtree (*combine*).

Berikut adalah pseudocode kedua prosedur tersebut.

```
procedure BuildQuadtree(node : Node, image: BufferedImage, errorMethod :  
String, level : integer)  
    // Mengecek apakah Node sudah merupakan daun atau belum  
    if node = NIL then  
        {do nothing}  
    else if node.width <= 1 and node.height <= 1 then
```

```

        node.isLeaf ← true
        leafCount ← leafCount + 1
    else if (node.width/2 * node.height/2) < minSize then
        node.isLeaf ← true
        leafCount ← leafCount + 1
    else if node.error <= threshold then
        node.isLeaf ← true
        leafCount ← leafCount + 1
    else // Membangun node-node anak
        node.isLeaf ← false
        if (level > depth) then
            depth ← level
        node.children ← new Node[4]
        i traversal [0..3] // Isi data node anak
            averageColor ← getAverageColor(node.x, node.y, position)
            colorNode ← new Color(averageColor)
            error ← calcError(node.x, node.y, position, errorMethod)
            node.children[i] ← new Node(node.x, node.y, position,
                colorNode, level, error)
        i traversal [0..3] // Build untuk node anak secara rekursif
            if node.children[i] ≠ NIL then
                BuildQuadtree(node.children[i], image, errorMethod,
                    level+1)

procedure FillImageData(node : Node, newImage : BufferedImage)
    if node = NIL then
        {do nothing}
    else if node.isLeaf = true then
        i traversal [(node.x)..(node.x+width)]
            j traversal [(node.y)..(node.y+height)]
                newImage.setRGB(i, j, node.argb)
    else
        for (Node child : node.children) do
            FillImageData(child, newImage)
        endfor

```

BAB II

IMPLEMENTASI

2.1. Implementasi Node.java

Kelas Node merupakan kelas yang berisi atribut-atribut sebuah *node* pada Quadtree. Atribut-atribut yang dimiliki kelas ini adalah:

- Posisi piksel paling kiri atas (x, y)
- Dimensi Node (*width, height*)
- Nilai warna rata-rata Node (*argb*)
- Tingkat kedalaman Node pada Quadtree (*level*)
- Error Node yang telah dihitung (*error*)
- Status apakah Node merupakan daun atau tidak (*isLeaf*)
- Array Node yang berisi Node-Node anak (*children*)

Fungsi/Prosedur/Class	Deskripsi
<pre>public class Node { int x, y; int width, height; int argb; int level; double error; boolean isleaf; Node[] children; }</pre>	Kelas Node berisi atribut-atributnya
<pre>public Node(int x, int y, int width, int height, int argb, int level, double error)</pre>	Konstruktor sebuah objek Node

2.2. Implementasi Quadtree.java

Kelas Quadtree merupakan kelas yang bertanggungjawab untuk membangun dan menghasilkan suatu Quadtree dari gambar yang diinginkan. Kelas ini memiliki atribut-atribut sebagai berikut.

- Akar berupa Node (*root*)

- Ambang batas untuk error (*threshold*)
- Ukuran minimum piksel yang diperbolehkan (*minSize*)
- Gambar dalam bentuk BufferedImage (*imageData*)
- Jumlah *node* yang dihasilkan (*nodeCount*)
- Jumlah daun yang dihasilkan (*leafCount*)
- Kedalaman pohon (*depth*)
- [Debugging] Perhitungan verbose (*verboseCount*)

Fungsi/Prosedur/Class	Deskripsi
<pre>public class Quadtree { public Node root; public double threshold; public int minSize; public BufferedImage imageData; private long nodeCount = 0; private long leafCount = 0; private int depth = 0; private long verboseCount = 0; }</pre>	Kelas Quadtree berisi atribut-atributnya
<pre>public Quadtree(double threshold, int minSize)</pre>	Konstruktor umum objek Quadtree
<pre>public void CreateQuadtree(BufferedImage imageData, String errorMethod, boolean verbose)</pre>	Prosedur untuk membuat Quadtree. Berisi inisialisasi Node akar Quadtree berdasarkan data dari citra dan melaksanakan pembangunan Quadtree melalui prosedur pembantu <i>BuildQuadtree</i> .
<pre>private void BuildQuadTree(Node node, BufferedImage imageData, String errorMethod, int level, boolean verbose)</pre>	Prosedur pembantu dalam pembangunan Quadtree secara rekursif dengan membagi Node menjadi empat sub-blok berdasarkan error yang telah dihitung. Node anak akan memanggil prosedur ini seterusnya hingga Node tidak dapat dibagi lagi.
<pre>public BufferedImage</pre>	Fungsi yang mengembalikan citra dalam bentuk BufferedImage dari Quadtree. Memanggil prosedur

<code>ImageFromQuadtree(String extension)</code>	pembantu <i>FillImageData</i> .
<code>public void FillImageData(Node node, BufferedImage newImageData)</code>	Prosedur yang mengisi gambar <i>newImageData</i> berdasarkan daun-daun pada Quadtree yang telah dibangun. Proses ini dilakukan secara rekursif untuk setiap Node anak hingga mencapai Node yang merupakan daun.
<code>public double calcError(int x, int y, int width, int height, String method)</code>	Fungsi untuk melakukan perhitungan error pada suatu blok gambar berdasarkan metode perhitungan error yang diinginkan.
<code>private void RefreshLeaves(Node node, double threshold)</code>	[BONUS] Prosedur untuk memperbarui status daun setiap Node pada Quadtree. Digunakan untuk mengerjakan bonus target persentase kompresi.
<code>public static Quadtree TargetedPercentageCompress(File originalImageFile, double targetCompressionPercentage, String errorMethod, double maxThresh, String extension, boolean verbose)</code>	[BONUS] Fungsi utama untuk bonus target persentase kompresi. Fungsi ini mengembalikan Quadtree berdasarkan persentase kompresi yang diinginkan oleh pengguna. Dengan menggunakan pencarian biner, nilai ambang batas disesuaikan supaya dapat menghasilkan gambar dengan persentase kompresi yang diinginkan.
<code>private void RefreshLeaves(Node node, int maxDepth)</code>	[BONUS] Prosedur untuk memperbarui status daun setiap Node pada Quadtree. Digunakan untuk mengerjakan bonus GIF.
<code>public BufferedImage[] GetFrames(String extension)</code>	[BONUS] Fungsi yang mengembalikan sebuah larik berisi BufferedImage untuk setiap tingkat kedalaman Quadtree (dibatasi maksimal 8 <i>frame</i> dengan <i>frame</i> terakhir adalah hasil akhir kompresi)

2.3. Implementasi IOHandler.java

Kelas IOHandler merupakan kelas pembantu untuk menangani *input* dan *output*, terutama dalam bentuk memvalidasi *input* dan melakukan *export* gambar/GIF yang sudah terkompresi.

Fungsi/Prosedur/Class	Deskripsi
<code>public static BufferedImage</code>	Fungsi yang mengembalikan gambar dalam bentuk

<code>getImage(String imagePath)</code>	BufferedImage berdasarkan alamat gambar.
<code>public static String getExtension(String imagePath)</code>	Fungsi yang mengembalikan ekstensi gambar berdasarkan alamatnya.
<code>public static long getFileSize(String imagePath)</code>	Fungsi yang mengembalikan ukuran file berdasarkan alamatnya.
<code>public static void saveImage(BufferedImage image, String absolutePath, String extension)</code>	Prosedur yang meng-export gambar hasil kompresi dan menyimpannya pada alamat yang telah ditentukan.
<code>public static String getOutputPath()</code>	Fungsi yang memvalidasi <i>input</i> alamat penyimpanan gambar setelah dikompresi.
<code>public static String getPathGIF()</code>	[BONUS] Fungsi yang memvalidasi <i>input</i> alamat penyimpanan GIF pembentukan gambar setelah dikompresi.
<code>public static String getFileName(String outputPath, String extension)</code>	Fungsi yang mengembalikan nama file gambar hasil kompresi.
<code>public static void createGIF(BufferedImage[] frames, String outputPath, String fileName)</code>	[BONUS] Prosedur untuk membuat GIF berdasarkan larik BufferedImage dan menyimpannya pada alamat yang diberikan.
<code>public static IIOMetadata setFrameMetadata(IIOMetadata metadata, int delayTime)</code>	[BONUS] Fungsi yang mengembalikan metadata <i>frame</i> GIF, termasuk pengaturan waktu tunda dan pengaturan loop GIF.
<code>private static IIOMetadataNode getNode(IIOMetadataNode root, String nodeName)</code>	[BONUS] Fungsi pembantu yang mengembalikan <i>node</i> metadata.

2.4. Implementasi Main.java

Kelas Main berisi driver utama menjalankan program. Fungsi ini berisi permintaan *input-input* dari pengguna dan memvalidasi masukannya, pencatatan setiap

masukan, pelaksanaan algoritma kompresi gambar, dan penyimpanan hasil kompresi ke dalam alamat yang telah dimasukkan.

BAB III

SOURCE CODE

3.1. Repository Program

Tautan menuju repository program adalah sebagai berikut:
https://github.com/TukangLas21/Tucil2_13523093_13523112

3.2. *Source Code* Program

3.2.1. Node.java

```
package quadtree;
public class Node {
    int x, y; // x as column, y as row
    int width, height;
    int argb;
    int level;
    double error;
    boolean isLeaf;
    Node[] children;

    public Node(int x, int y, int width, int height, int argb, int
level, double error) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.argb = argb;
        this.level = level;
        this.error = error;
        this.isLeaf = true; // Set isLeaf to true by default
        this.children = null;
    }
}
```

3.2.2. Quadtree.java

```
package quadtree;
import java.awt.Color;
import java.awt.image.BufferedImage;
```

```

import java.io.File;
import java.io.IOException;
import java.nio.file.Files;
import java.nio.file.Path;
import java.nio.file.Paths;
import java.util.Comparator;

import javax.imageio.ImageIO;

public class Quadtree {
    public Node root;
    public double threshold; // minimum 0.0, Double.NEGATIVE_INFINITY
    to guarantee maximum depth
    public int minSize; // minimum 1 pixel
    public BufferedImage imageData;
    private long nodeCount = 0;
    private long leafCount = 0;
    private int depth = 0;
    private long verboseCount = 0;

    public Quadtree(double threshold, int minSize) {
        this.root = null;
        this.threshold = threshold;
        this.minSize = minSize;
        this.imageData = null;
    }

    public Quadtree() {
        this.root = null;
        this.threshold = Double.NEGATIVE_INFINITY; // maximum depth
        this.minSize = 1;
        this.imageData = null;
    }

    public long getNodeCount() {
        return nodeCount;
    }

    public int getDepth() {

```

```

        return depth;
    }

    public long getLeafCount() {
        return leafCount;
    }

    public BufferedImage getImageData() {
        return imageData;
    }

    public void CreateQuadtree(BufferedImage imageData, String
errorMethod, boolean verbose) {
        if (imageData == null) {
            throw new IllegalArgumentException("Image data cannot be
null.");
        }
        this.imageData = imageData;
        int width = imageData.getWidth();
        int height = imageData.getHeight();
        int avgRed = (int) getAverage(0, 0, width, height, "r");
        int avgGreen = (int) getAverage(0, 0, width, height, "g");
        int avgBlue = (int) getAverage(0, 0, width, height, "b");
        Color colorNode = new Color(avgRed, avgGreen, avgBlue);
        double error = calcError(0, 0, width, height, errorMethod);
        this.root = new Node(0, 0, width, height, colorNode.getRGB(),
0, error);
        this.leafCount = 0;
        this.nodeCount = 1;
        this.depth = 0;
        this.verboseCount = 0;

        BuildQuadTree(root, imageData, errorMethod, 1, verbose);
    }

    public void CreateQuadtree(BufferedImage imageData, String
errorMethod) {
        CreateQuadtree(imageData, errorMethod, false);
    }

```

```

// Helper method to build the quadtree recursively
private void BuildQuadTree(Node node, BufferedImage imageData,
String errorMethod, int level, boolean verbose) {

    if (node == null) {
        return;
    }
    if (node.width <= 1 && node.height <= 1) {
        node.isLeaf = true;
        leafCount++;
        return;
    }
    if ((node.width/2 * node.height/2) < minSize) {
        node.isLeaf = true;
        leafCount++;
        return;
    }
    if (node.error <= threshold) {
        node.isLeaf = true;
        leafCount++;
        return;
    }

    // Split the node into 4 children
    node.isLeaf = false;

    if (level > depth) {
        depth = level;
    }

    // Verbose output for debugging
    if (verbose) {
        if (level < 5) System.out.println("[BUILDING QUADTREE] ID
" + verboseCount++ + ": Node Count: " + nodeCount + ", Depth: " +
depth + ", Level: " + level + ", Error: " + node.error);
    }

    int leftHalfWidth = (node.width + 1) / 2;

```

```

    int topHalfHeight = (node.height + 1) / 2;
    int rightHalfWidth = node.width - leftHalfWidth;
    int bottomHalfHeight = node.height - topHalfHeight;

    node.children = new Node[4];

    int avgRed = (int) getAverage(node.x, node.y, leftHalfWidth,
topHalfHeight, "r");
    int avgGreen = (int) getAverage(node.x, node.y,
leftHalfWidth, topHalfHeight, "g");
    int avgBlue = (int) getAverage(node.x, node.y, leftHalfWidth,
topHalfHeight, "b");
    Color colorNode = new Color(avgRed, avgGreen, avgBlue);
    int argb = colorNode.getRGB();
    double error = calcError(node.x, node.y, leftHalfWidth,
topHalfHeight, errorMethod); // Kiri Atas

    node.children[0] = new Node(node.x, node.y, leftHalfWidth,
topHalfHeight, argb, level, error); // Kiri Atas

    avgRed = (int) getAverage(node.x + leftHalfWidth, node.y,
rightHalfWidth, topHalfHeight, "r");
    avgGreen = (int) getAverage(node.x + leftHalfWidth, node.y,
rightHalfWidth, topHalfHeight, "g");
    avgBlue = (int) getAverage(node.x + leftHalfWidth, node.y,
rightHalfWidth, topHalfHeight, "b");
    colorNode = new Color(avgRed, avgGreen, avgBlue);
    argb = colorNode.getRGB();
    error = calcError(node.x + leftHalfWidth, node.y,
rightHalfWidth, topHalfHeight, errorMethod); // Kanan Atas

    node.children[1] = new Node(node.x + leftHalfWidth, node.y,
rightHalfWidth, topHalfHeight, argb, level, error); // Kanan Atas

    avgRed = (int) getAverage(node.x, node.y + topHalfHeight,
leftHalfWidth, bottomHalfHeight, "r");
    avgGreen = (int) getAverage(node.x, node.y + topHalfHeight,
leftHalfWidth, bottomHalfHeight, "g");
    avgBlue = (int) getAverage(node.x, node.y + topHalfHeight,

```



```

leftHalfWidth, bottomHalfHeight, "b");
    colorNode = new Color(avgRed, avgGreen, avgBlue);
    argb = colorNode.getRGB();
    error = calcError(node.x, node.y + topHalfHeight,
leftHalfWidth, bottomHalfHeight, errorMethod); // Kiri Bawah

    node.children[2] = new Node(node.x, node.y + topHalfHeight,
leftHalfWidth, bottomHalfHeight, argb, level, error); // Kiri Bawah

    avgRed = (int) getAverage(node.x + leftHalfWidth, node.y +
topHalfHeight, rightHalfWidth, bottomHalfHeight, "r");
    avgGreen = (int) getAverage(node.x + leftHalfWidth, node.y +
topHalfHeight, rightHalfWidth, bottomHalfHeight, "g");
    avgBlue = (int) getAverage(node.x + leftHalfWidth, node.y +
topHalfHeight, rightHalfWidth, bottomHalfHeight, "b");
    colorNode = new Color(avgRed, avgGreen, avgBlue);
    argb = colorNode.getRGB();
    error = calcError(node.x + leftHalfWidth, node.y +
topHalfHeight, rightHalfWidth, bottomHalfHeight, errorMethod); //
Kanan Bawah

    node.children[3] = new Node(node.x + leftHalfWidth, node.y +
topHalfHeight, rightHalfWidth, bottomHalfHeight, argb, level, error);
// Kanan Bawah

    for (int i = 0; i < 4; i++) {
        if (node.children[i] != null) {
            BuildQuadTree(node.children[i], imageData,
errorMethod, level + 1, verbose);
            nodeCount++;
        }
    }
}

public BufferedImage ImageFromQuadtree(String extension) {
    if (extension.equals("png") || extension.equals("gif") ||
extension.equals("tiff")) {
        BufferedImage newImageData = new
BufferedImage(root.width, root.height, BufferedImage.TYPE_INT_ARGB);

```

```

        FillImageData(root, newImageData);
        return newImageData;
    } else {
        BufferedImage newImageData = new
BufferedImage(root.width, root.height, BufferedImage.TYPE_INT_RGB);
        FillImageData(root, newImageData);
        return newImageData;
    }
}

public void FillImageData(Node node, BufferedImage newImageData)
{
    if (node == null) {
        return;
    }
    if (node.isLeaf) {
        for (int i = node.x; i < node.x + node.width; i++) {
            for (int j = node.y; j < node.y + node.height; j++) {
                newImageData.setRGB(i, j, (node.argb &
0x00FFFFFF) | (imageData.getRGB(i, j) & 0xFF000000));
            }
        }
        return;
    }
    for (Node child : node.children) {
        FillImageData(child, newImageData);
    }
}

public void RefreshLeaves(double threshold) {
    if (root == null) {
        throw new IllegalStateException("Quadtree is not
initialized. Please call CreateQuadtree() first.");
    }
    this.threshold = threshold;
    this.leafCount = 0;
    RefreshLeaves(root, threshold);
}

```

```

public void RefreshLeaves(int maxDepth) {
    if (root == null) {
        throw new IllegalStateException("Quadtree is not
initialized. Please call CreateQuadtree() first.");
    }
    this.leafCount = 0;
    RefreshLeaves(root, maxDepth);
}

private void RefreshLeaves(Node node, double threshold) {
    if (node == null) {
        return;
    }
    if (node.children == null) {
        node.isLeaf = true;
        leafCount++;
        return;
    }
    if (node.error <= threshold) {
        node.isLeaf = true;
        leafCount++;
        return;
    }
    node.isLeaf = false;
    for (Node child : node.children) {
        RefreshLeaves(child, threshold);
    }
}

private void RefreshLeaves(Node node, int maxDepth) {
    if (node == null) {
        return;
    }
    if (node.children == null) {
        node.isLeaf = true;
        leafCount++;
        return;
    }
    if (node.level == maxDepth) {

```

```

        node.isLeaf = true;
        leafCount++;
        return;
    }
    node.isLeaf = false;
    for (Node child : node.children) {
        RefreshLeaves(child, maxDepth);
    }
}

public static double MaximumError(String errorMethod) {
    return switch (errorMethod) {
        case "variance" -> 16256.25; // Variance threshold for
8-bit color depth (255^2 / 4)
        case "mad" -> 127.5; // Maximum absolute difference for
8-bit color depth (255 / 2)
        case "mpd" -> 255; // Maximum possible difference for
8-bit color depth (255 - 0)
        case "entropy" -> 8.0; // Maximum entropy for 8-bit color
depth (log2(256))
        default -> throw new IllegalArgumentException("Invalid
error method: " + errorMethod);
    };
}

public void RefreshLeaves() {
    if (root == null) {
        throw new IllegalStateException("Quadtree is not
initialized. Please call CreateQuadtree() first.");
    }
    this.leafCount = 0;
    RefreshLeaves(root, threshold);
}

public BufferedImage[] GetFrames(String extension) {
    int maxDepth = 8;
    if (this.depth < maxDepth) {
        BufferedImage[] frames = new BufferedImage[this.depth +
1];

```

```

        for (int i = 0; i <= this.depth; i++) {
            RefreshLeaves(i);
            frames[i] = ImageFromQuadtree(extension);
        }
        return frames;

    } else {
        BufferedImage[] frames = new BufferedImage[maxDepth];
        for (int i = 0; i < maxDepth-1; i++) {
            RefreshLeaves(i);
            frames[i] = ImageFromQuadtree(extension);
        }
        RefreshLeaves(this.depth);
        frames[maxDepth-1] = ImageFromQuadtree(extension); // Set
the last frame to the maximum depth image
        return frames;
    }
}

/**
 *
 * @param originalImageFile
 * @param targetCompressionPercentage Range 0.0 - 100.0
 * @param errorMethod
 * @return
 */
public static Quadtree TargetedPercentageCompress(File
originalImageFile, double targetCompressionPercentage, String
errorMethod, double maxThresh, String extension, boolean verbose) {
    if (targetCompressionPercentage < 0 ||
targetCompressionPercentage > 100) {
        throw new IllegalArgumentException("Target compression
percentage must be between 0 and 100.");
    }

    // Create bin folder to store temporary image file
    String binDir = System.getProperty("java.io.tmpdir") +
File.separator + "quadtree" + File.separator + "bin";
    File binFolder = new File(binDir);

```

```

        if (!binFolder.exists()) {
            binFolder.mkdirs();
        }

        // Initialize image
        BufferedImage image;
        try {
            image = ImageIO.read(originalImageFile);
        } catch (Exception e) {
            throw new RuntimeException("Error reading image file: " +
e.getMessage());
        }

        if (image == null) {
            throw new IllegalArgumentException("Image not found.
Please try again.");
        }

        // Get original file size
        long originalFileSize = originalImageFile.length();

        // Initialize quadtree parameters
        // double upper = 255 * 255 / 4;
        double upper = maxThresh;
        double lower = 0;
        double mid = (upper + lower) / 2;
        double tolerance = 1; // Tolerance for binary search
        double stoppingThreshold = 0.5; // Stopping threshold for
binary search
        int maxStoppingThresholdCount = 5;

        int stoppingThresholdCount = maxStoppingThresholdCount;
        int iteration = 1;
        double lastPercent = targetCompressionPercentage;
        File compressedImageFile;

        // Initialize and build quadtree with maximum depth and node
count
        Quadtree quadtree = new Quadtree();

```

```

        quadtree.CreateQuadtree(image, errorMethod, verbose);

        while (true) {
            // Create quadtree with current threshold
            quadtree.RefreshLeaves(mid);
            BufferedImage compressedImage =
quadtree.ImageFromQuadtree(extension);
            compressedImageFile = new File(binDir + File.separator +
"compressed_image." + extension);
            try {
                ImageIO.write(compressedImage, extension,
compressedImageFile);
            } catch (Exception e) {
                throw new RuntimeException("Error writing compressed
image file: " + e.getMessage());
            }
            long compressedFileSize = compressedImageFile.length();
            double percent = compressionPercentage(originalFileSize,
compressedFileSize);

            // Verbose
            if (verbose) {
                System.out.println("[TARGETED COMPRESSION] Iteration
" + iteration + ": Compression percentage: " + percent + "%, Target:
" + targetCompressionPercentage + "%, File size: " +
compressedFileSize + " bytes, Threshold: " + mid);
            }
            iteration++;

            if (Math.abs(percent - lastPercent) < stoppingThreshold)
{
                stoppingThresholdCount--;
                if (stoppingThresholdCount <= 0) {
                    try {
                        deleteTempBin();
                    } catch (IOException e) {
                        System.err.println("Error deleting temporary
bin: " + e.getMessage());
                    }
                }
            }
        }
    }
}

```

```

        return quadtree;
    }
} else {
    stoppingThresholdCount = maxStoppingThresholdCount;
// Reset if the difference is significant
}

lastPercent = percent;

if (percent >= targetCompressionPercentage) {
    if (Math.abs(percent - targetCompressionPercentage)
<= tolerance) {
        try {
            deleteTempBin();
        } catch (IOException e) {
            System.err.println("Error deleting temporary
bin: " + e.getMessage());
        }
        return quadtree;
    } else {
        upper = mid;
    }
} else {
    lower = mid;
}
mid = (upper + lower) / 2;
}
}

public static void deleteTempBin() throws IOException {
    Path binDir = Paths.get(System.getProperty("java.io.tmpdir"),
"quadtree", "bin");
    if (Files.exists(binDir)) {
        Files.walk(binDir)
            .sorted(Comparator.reverseOrder())
            .forEach(path -> {
                try { Files.delete(path); }
                catch (IOException e) {
                    System.err.println("Failed to delete file: "

```



```

+ path + " - " + e.getMessage());
        }
    });
}

    public double calcError(int x, int y, int width, int height,
String method) {
        double error = Double.POSITIVE_INFINITY;
        switch (method) {
            case "variance" -> error = calcVariance(x, y, width,
height);
            case "mad" -> error = calcMAD(x, y, width, height);
            case "mpd" -> error = calcMPD(x, y, width, height);
            case "entropy" -> error = calcEntropy(x, y, width,
height);
            // case "ssim" -> error = calcSSIM(x, y, width, height);
            default -> {
                throw new IllegalArgumentException("Invalid error
method: " + method);
            }
        }
        return error;
    }

    // Variansi
    public double calcVariance(int x, int y, int width, int height) {
        double varRed = getVariance(x, y, width, height, "r");
        double varGreen = getVariance(x, y, width, height, "g");
        double varBlue = getVariance(x, y, width, height, "b");
        // double varAlpha = getVariance(x, y, width, height, "a");

        return (varRed + varGreen + varBlue) / 3.0;
    }

    // Mendapatkan nilai rata-rata setiap kanal
    public double getAverage(int x, int y, int width, int height,
String channel) {

```

```

        double sum = 0;
        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                switch (channel) {
                    case "r" -> sum += (imageData.getRGB(i, j) >> 16)
& 0xFF; // Mengambil nilai merah dari argb
                    case "g" -> sum += (imageData.getRGB(i, j) >> 8)
& 0xFF; // Mengambil nilai hijau dari argb
                    case "b" -> sum += imageData.getRGB(i, j) & 0xFF;
// Mengambil nilai biru dari argb
                    case "a" -> sum += (imageData.getRGB(i, j) >> 24)
& 0xFF; // Mengambil nilai alpha dari argb
                }
            }
        }
        int length = width * height;
        return sum / (length == 0 ? 1 : length); // Menghindari
pembagian dengan nol
    }

```

```

// Mendapatkan nilai variansi setiap kanal warna
public double getVariance(int x, int y, int width, int height,
String channel) {
    double avg = getAverage(x, y, width, height, channel);
    double sum = 0;
    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            switch (channel) {
                case "r" -> sum += Math.pow(((imageData.getRGB(i,
j) >> 16) & 0xFF) - avg, 2); // Mengambil nilai merah dari argb
                case "g" -> sum += Math.pow(((imageData.getRGB(i,
j) >> 8) & 0xFF) - avg, 2); // Mengambil nilai hijau dari argb
                case "b" -> sum += Math.pow((imageData.getRGB(i,
j) & 0xFF) - avg, 2); // Mengambil nilai biru dari argb
                case "a" -> sum += Math.pow(((imageData.getRGB(i,
j) >> 24) & 0xFF) - avg, 2); // Mengambil nilai alpha dari argb
            }
        }
    }
}

```

```

    }
    int length = width * height;
    return sum / (length == 0 ? 1 : length); // Menghindari
pembagian dengan nol
}

public double calcMAD(int x, int y, int width, int height) {
    double madRed = getMAD(x, y, width, height, "r");
    double madGreen = getMAD(x, y, width, height, "g");
    double madBlue = getMAD(x, y, width, height, "b");
    // double madAlpha = getMAD(x, y, width, height, "a");

    return (madRed + madGreen + madBlue) / 3.0;
}

// Mendapatkan nilai MAD setiap kanal warna
public double getMAD(int x, int y, int width, int height, String
channel) {
    double avg = getAverage(x, y, width, height, channel);
    double sum = 0;
    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            switch (channel) {
                case "r" -> sum += Math.abs(((imageData.getRGB(i,
j) >> 16) & 0xFF) - avg);
                case "g" -> sum += Math.abs(((imageData.getRGB(i,
j) >> 8) & 0xFF) - avg);
                case "b" -> sum += Math.abs((imageData.getRGB(i,
j) & 0xFF) - avg);
                case "a" -> sum += Math.abs(((imageData.getRGB(i,
j) >> 24) & 0xFF) - avg);
            }
        }
    }
    int length = width * height;
    return sum / (length == 0 ? 1 : length);
}

public double calcMPD(int x, int y, int width, int height) {

```

```

        double mpdRed = getMPD(x, y, width, height, "r");
        double mpdGreen = getMPD(x, y, width, height, "g");
        double mpdBlue = getMPD(x, y, width, height, "b");
        // double mpdAlpha = getMPD(x, y, width, height, "a");

        return (mpdRed + mpdGreen + mpdBlue) / 3.0;
    }

    public double getMPD(int x, int y, int width, int height, String
channel) {
        int min = Integer.MAX_VALUE;
        int max = Integer.MIN_VALUE;
        int value = 0;

        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                switch (channel) {
                    case "r" -> value = ((imageData.getRGB(i, j) >>
16) & 0xFF);
                    case "g" -> value = ((imageData.getRGB(i, j) >>
8) & 0xFF);
                    case "b" -> value = (imageData.getRGB(i, j) &
0xFF);
                    case "a" -> value = ((imageData.getRGB(i, j) >>
24) & 0xFF);
                }
                if (value < min) {
                    min = value;
                }
                if (value > max) {
                    max = value;
                }
            }
        }
        return max - min;
    }

    public double calcEntropy(int x, int y, int width, int height) {
        double entropyRed = getEntropy(x, y, width, height, "r");

```

```

        double entropyGreen = getEntropy(x, y, width, height, "g");
        double entropyBlue = getEntropy(x, y, width, height, "b");
        // double entropyAlpha = getEntropy(x, y, width, height,
"a");

        return (entropyRed + entropyGreen + entropyBlue) / 3.0;
    }

    public double getEntropy(int x, int y, int width, int height,
String channel) {
        double entropy = 0;
        int[] histogram = new int[256];
        int totalPixels = width * height;

        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                switch (channel) {
                    case "r" -> histogram[(imageData.getRGB(i, j) >>
16) & 0xFF]++;
                    case "g" -> histogram[(imageData.getRGB(i, j) >>
8) & 0xFF]++;
                    case "b" -> histogram[imageData.getRGB(i, j) &
0xFF]++;
                    case "a" -> histogram[(imageData.getRGB(i, j) >>
24) & 0xFF]++;
                }
            }
        }

        for (int i = 0; i < histogram.length; i++) {
            if (histogram[i] > 0) {
                double p = (double) histogram[i] / totalPixels;
                entropy -= p * Math.log(p) / Math.log(2);
            }
        }

        return entropy;
    }

```

```

        public static double compressionPercentage(long originalSize,
long compressedSize) {
            return ((double) (originalSize - compressedSize) /
originalSize) * 100;
        }
    }
}

```

3.2.3. IOHandler.java

```

package quadtree;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;

import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageTypeSpecifier;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOInvalidTreeException;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.stream.ImageOutputStream;

import org.w3c.dom.NodeList;
public class IOHandler {

    private static Scanner scanner = new Scanner(System.in);

    // Load image dari path sebagai BufferedImage
    public static BufferedImage getImage(String imagePath) {
        try {
            File imageFile = new File(imagePath);
            BufferedImage image = ImageIO.read(imageFile);
            return image;
        } catch (IOException e) {
            System.err.println("Error loading image: " +
e.getMessage());

```

```

        return null;
    }
}

// Mendapatkan file extension
public static String getExtension(String imagePath) {
    int dotIndex = imagePath.lastIndexOf(".");
    if (dotIndex > 0 && dotIndex < imagePath.length() - 1) {
        return imagePath.substring(dotIndex + 1).toLowerCase();
    }
    return ""; // Return string kosong jika tidak ada extension
}

// Mendapatkan ukuran file
public static long getFileSize(String imagePath) {
    File file = new File(imagePath);
    if (file.exists()) {
        return file.length(); // Mengembalikan ukuran file dalam
byte
    } else {
        System.err.println("File not found: " + imagePath);
        return -1; // Mengembalikan -1 jika file tidak ditemukan
    }
}

// Save image pada output path
public static void saveImage(BufferedImage image, String
absolutePath, String extension) {
    try {
        File outputFile = new File(absolutePath);

        // Save the image
        boolean isSaved = ImageIO.write(image, extension,
outputFile);
        if (isSaved) {
            System.out.println("Gambar kompresi berhasil disimpan
di " + outputFile.getAbsolutePath());
        } else {
            System.err.println("Error saving image.");
        }
    }
}

```

```

    }
    } catch (IOException e) {
        System.err.println("Error saving image: " +
e.getMessage());
    }
}

// Mendapatkan dan validasi input output path
public static String getOutputPath() {
    System.out.print("Silakan masukkan alamat absolut output
(tanpa nama): ");
    String outputPath =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");

    while (true) {
        File path = new File(outputPath);

        if (path.exists() && path.isDirectory()) {
            return outputPath;
        } else {
            System.out.println("Alamat yang dimasukkan tidak
valid.");
            System.out.print("Silakan masukkan alamat absolut
output (tanpa nama): ");
            outputPath =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");
        }
    }
}

// Mendapatkan dan validasi input path GIF
public static String getPathGIF() {
    System.out.print("Silakan masukkan alamat GIF absolut (tanpa
nama file): ");
    String gifPath =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");

    while (true) {
        File path = new File(gifPath);

```



```

        if (path.exists() && path.isDirectory()) {
            return gifPath;
        } else {
            System.out.println("Alamat yang dimasukkan tidak
valid.");
            System.out.print("Silakan masukkan alamat GIF absolut
(tanpa nama file): ");
            gifPath =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");
        }
    }
}

// Method untuk mendapatkan nama file output sekaligus
memvalidasikan input pengguna
public static String getFileName(String outputPath, String
extension) {
    System.out.print("Silakan masukkan nama file (tanpa
ekstensi): ");
    String fileName =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");
    File filePath = new File(outputPath, fileName + "." +
extension);

    // Cek apakah sudah ada file dengan nama yang sama
    while (filePath.exists() || fileName.equals("")) {
        System.out.println("File sudah ada atau masukan tidak
valid. Silakan coba lagi.");
        System.out.print("Masukkan nama file (tanpa ekstensi):
");
        fileName =
scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");
        filePath = new File(outputPath, fileName + "." +
extension);
    }

    return fileName;
}

```

```

    // Membuat GIF dari array BufferedImage
    public static void createGIF(BufferedImage[] frames, String
outputPath, String fileName) {
        try {
            ImageWriter writer =
ImageIO.getImageWritersByFormatName("gif").next();
            ImageOutputStream ios =
ImageIO.createImageOutputStream(new File(outputPath, fileName +
".gif"));
            writer.setOutput(ios);

            writer.prepareWriteSequence(null);

            int delayTime = 100; // Jeda waktu setiap frame 1 detik

            // Write frame ke dalam GIF
            for (BufferedImage frame : frames) {
                ImageWriteParam params =
writer.getDefaultWriteParam();
                IIOMetadata metadata =
writer.getDefaultImageMetadata(new ImageTypeSpecifier(frame),
params);
                metadata = setFrameMetadata(metadata, delayTime);
                IIOWrite tempFrame = new IIOWrite(frame, null,
metadata);
                writer.writeToSequence(tempFrame, params);
            }

            writer.endWriteSequence();
            ios.close();
            writer.dispose();

        } catch (IOException e) {
            System.err.println("Error creating GIF: " +
e.getMessage());
        }
    }
}

```

```

    public static IIOMetadata setFrameMetadata(IIOMetadata metadata,
int delayTime) throws IIOMetadataException {
        String metaFormatName =
metadata.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode)
metadata.getAsTree(metaFormatName);

        // Node Graphic Control Extension
        IIOMetadataNode graphicsControlExtensionNode = getNode(root,
"GraphicControlExtension");
        if (graphicsControlExtensionNode == null) {
            graphicsControlExtensionNode = new
IIOMetadataNode("GraphicControlExtension");
            root.appendChild(graphicsControlExtensionNode);
        }

        // Set attribute untuk node GraphicControlExtension secara
manual
        graphicsControlExtensionNode.setAttribute("disposalMethod",
"none");
        graphicsControlExtensionNode.setAttribute("userInputFlag",
"FALSE");

        graphicsControlExtensionNode.setAttribute("transparentColorFlag",
"FALSE");
        graphicsControlExtensionNode.setAttribute("delayTime",
Integer.toString(delayTime)); // set delay time here

        graphicsControlExtensionNode.setAttribute("transparentColorIndex",
"0");

        // Node Application Extensions
        IIOMetadataNode appExtensionsNode = getNode(root,
"ApplicationExtensions");

        if (appExtensionsNode == null) { // Jika tidak ada, dibuat
node baru
            appExtensionsNode = new
IIOMetadataNode("ApplicationExtensions");

```

```

        root.appendChild(appExtensionsNode);
    }

    IIOMetadataNode appExtension = new
IIOMetadataNode("ApplicationExtension");

    // Set attribute untuk node ApplicationExtension
    appExtension.setAttribute("applicationID", "NETSCAPE");
    appExtension.setAttribute("authenticationCode", "2.0");

    // Loop selamanya
    byte[] appExtensionBytes = new byte[] { 0x1, 0x0, 0x0 };
    appExtension.setUserObject(appExtensionBytes);
    appExtensionsNode.appendChild(appExtension);

    metadata.setFromTree(metaFormatName, root);
    return metadata;
}

private static IIOMetadataNode getNode(IIOMetadataNode root,
String nodeName) {
    NodeList nodeList = root.getElementsByTagName(nodeName); //
Mencari node yang sesuai
    if (nodeList.getLength() > 0) {
        return (IIOMetadataNode) nodeList.item(0);
    }
    return null; // Mengembalikan null jika tidak ada
}

// Dapatkan image data dalam bentuk array 3D
// Array: [x][y][arrRGB]
public static int[][][] getImageData(BufferedImage image) {
    int width = image.getWidth();
    int height = image.getHeight();
    int[][][] imageData = new int[width][height][4];

    for (int i = 0; i < width; i++) {
        for (int j = 0; j < height; j++) {

```

```

        int rgb = image.getRGB(i, j);
        imageData[i][j] = extractRGB(rgb);
    }
}
return imageData;
}

// Extract nilai RGB dari sebuah piksel
// arrRGB = [Red, Green, Blue, Alpha]
public static int[] extractRGB(int rgb) {
    int[] arrRGB = new int[4];
    arrRGB[0] = (rgb >> 16) & 0xFF; // Red
    arrRGB[1] = (rgb >> 8) & 0xFF; // Green
    arrRGB[2] = rgb & 0xFF; // Blue
    arrRGB[3] = (rgb >> 24) & 0xFF; // Alpha
    return arrRGB;
}
}

```

3.2.4. Main.java

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.util.Scanner;

import quadtree.IOHandler;
import quadtree.Quadtree;

public class Main {

    private static Scanner scanner = new Scanner(System.in);

    public static void main(String[] args) {

System.out.println("-----");
        -----");
        System.out.println("Compressor QuadTree Image");
    }
}

```

```

System.out.println("-----
-----");

    while (true) {

        // Inputs
        // Input alamat absolut gambar
        System.out.print("Silakan masukkan alamat gambar, gunakan
        \"/\ " sebagai seperator: ");
        String filePath =
        scanner.nextLine().replaceAll("^['\\"]+|['\\"]+$", "");

        // Validasi gambar yang di-input
        BufferedImage image = IOHandler.getImage(filePath);
        if (image == null) {
            System.out.println("Gambar tidak ditemukan. Silakan
            coba lagi.");
            continue;
        }
        String extension = IOHandler.getExtension(filePath);
        if (extension.isEmpty()) {
            System.out.println("Format gambar tidak valid.
            Silakan coba lagi.");
            continue;
        }
        long fileSizeBefore = IOHandler.getFileSize(filePath); //
        Catat ukuran file sebelum dikompresi

        // Input metode perhitungan error
        String errorMethod = "";
        while (true) {
            System.out.println("Silakan pilih metode perhitungan
            error: ");

            System.out.println("1. Variance");
            System.out.println("2. Mean Absolute Deviation");
            System.out.println("3. Max Pixel Difference");
            System.out.println("4. Entropy");
            System.out.print("Pilihan: ");

```

```

        String userChoice = scanner.nextLine();
        if (userChoice.isEmpty()) {
            System.out.println("Pilihan tidak valid. Silakan
coba lagi.");
            continue;
        }
        int choice;
        try {
            choice = Integer.parseInt(userChoice);
        } catch (NumberFormatException e) {
            System.out.println("Pilihan tidak valid. Silakan
coba lagi.");
            continue;
        }

        if (choice < 1 || choice > 4) {
            System.out.println("Pilihan tidak valid. Silakan
coba lagi.");
            continue;
        }

        switch (choice) {
            case 1:
                errorMethod = "variance";
                break;
            case 2:
                errorMethod = "mad";
                break;
            case 3:
                errorMethod = "mpd";
                break;
            case 4:
                errorMethod = "entropy";
                break;
        }

        break;
    }

```

```

        // Input ambang batas
        String thresholdInput = "";
        double threshold = 0;
        while (true) {
            System.out.print("Silakan masukkan ambang batas (max:
" + Quadtree.MaximumError(errorMethod) + "): ");
            thresholdInput = scanner.nextLine();
            if (thresholdInput.isEmpty()) {
                System.out.println("Ambang batas tidak valid.
Silakan coba lagi.");
                continue;
            }
            try {
                threshold = Double.parseDouble(thresholdInput);
            } catch (NumberFormatException e) {
                System.out.println("Ambang batas tidak valid.
Silakan coba lagi.");
                continue;
            }
            if (threshold < 0 || threshold >
Quadtree.MaximumError(errorMethod)) {
                System.out.println("Ambang batas tidak valid.
Silakan coba lagi.");
                continue;
            }
            break;
        }

        // Input ukuran blok minimum
        String minBlockSizeInput = "";
        int minBlockSize = 0;
        while (true) {
            System.out.print("Silakan masukkan ukuran blok
minimum: ");
            minBlockSizeInput = scanner.nextLine();
            if (minBlockSizeInput.isEmpty()) {
                System.out.println("Ukuran blok minimum tidak
valid. Silakan coba lagi.");
                continue;
            }

```



```

        }
        try {
            minBlockSize =
Integer.parseInt(minBlockSizeInput);
        } catch (NumberFormatException e) {
            System.out.println("Ukuran blok minimum tidak
valid. Silakan coba lagi.");
            continue;
        }
        if (minBlockSize < 1) {
            System.out.println("Ukuran blok minimum tidak
valid. Silakan coba lagi.");
            continue;
        }
        break;
    }

    // Input target kompresi
    String targetKompresi = "";
    double targetPercent = 0;
    while (true) {
        System.out.print("Silakan masukkan target kompresi
(0..1): ");

        targetKompresi = scanner.nextLine();
        if (targetKompresi.isEmpty()) {
            System.out.println("Target kompresi tidak valid.
Silakan coba lagi.");
            continue;
        }
        try {
            targetPercent =
Double.parseDouble(targetKompresi);
        } catch (NumberFormatException e) {
            System.out.println("Target kompresi tidak valid.
Silakan coba lagi.");
            continue;
        }
        if (targetPercent < 0 || targetPercent > 1) {
            System.out.println("Masukkan target tidak valid.

```

```

        Silakan masukkan dalam bentuk floating number (0..1).");
    } else {
        break;
    }
}

targetPercent *= 100;

// Input alamat absolut gambar hasil kompresi
String outputPath = IOHandler.getOutputPath();

String outputFileName = IOHandler.getFileName(outputPath,
extension);

String absolutePath = outputPath + File.separator +
outputFileName + "." + extension;

// Input pilihan dan alamat absolut GIF
boolean saveGIF = false;
System.out.print("Apakah Anda ingin menyimpan dalam
bentuk GIF? (Y/N): ");
String gifChoice = scanner.nextLine();
while (true) {
    if (gifChoice.equals("y") || gifChoice.equals("Y")) {
        saveGIF = true;
        break;
    } else if (gifChoice.equals("n") ||
gifChoice.equals("N")) {
        break;
    } else {
        System.out.println("Masukan tidak valid, silakan
masukkan Y atau N: ");
        gifChoice = scanner.nextLine();
    }
}

String outputPathGIF = "";
String GIFname = "";
if (saveGIF) {

```

```

        outputPathGIF = IOHandler.getPathGIF();
        GIFname = IOHandler.getFileName(outputPathGIF,
"gif");
    }

    // Proses + catat waktu
    long startTime;
    Quadtree quadtree;
    if (targetPercent == 0.0) {
        startTime = System.currentTimeMillis();
        quadtree = new Quadtree(threshold, minBlockSize);
        System.out.println("Mengompresi gambar dengan metode
" + errorMethod + ", threshold " + threshold + ", dan minimum size "
+ minBlockSize + ".");
        quadtree.CreateQuadtree(image, errorMethod);
    } else {
        File imageFile = new File(filePath);
        System.out.println("Anda memilih untuk mengompresi
gambar dengan target persentase " + targetPercent + "%" + "
menggunakan metode " + errorMethod + " dan minimum size " +
minBlockSize + ".");
        System.out.print("Silakan pilih batas atas pencarian
threshold. Tekan enter untuk default (" +
Quadtree.MaximumError(errorMethod) + "): ");
        String maxThresholdInput;
        double maxThreshold = 0;
        while (true) {
            maxThresholdInput = scanner.nextLine();
            if (maxThresholdInput.isEmpty()) {
                break;
            }
            try {
                maxThreshold =
Double.parseDouble(maxThresholdInput);
            } catch (NumberFormatException e) {
                System.out.println("Ambang batas tidak valid.
Silakan coba lagi.");
                continue;
            }
        }
    }

```

```

        if (maxThreshold < 0) {
            System.out.println("Ambang batas tidak valid.
Silakan coba lagi.");
            continue;
        }
        break;
    }
    if (maxThresholdInput.isEmpty()) {
        maxThreshold =
Quadtree.MaximumError(errorMethod);
    }
    startTime = System.currentTimeMillis();
    System.out.println("Mengompresi gambar dengan target
persentase " + targetPercent + "%, metode " + errorMethod + ",
threshold " + threshold + ", minimum size " + minBlockSize + ", dan
batas atas pencarian threshold " + maxThreshold + ".");
    quadtree =
Quadtree.TargetedPercentageCompress(imageFile, targetPercent,
errorMethod, maxThreshold, extension, false);
    }
    long endTime = System.currentTimeMillis();
    long runTime = endTime - startTime;

    // Outputs
    // Output gambar setelah dikompresi
    BufferedImage compressedImage =
quadtree.ImageFromQuadtree(extension);
    if (compressedImage == null) {
        System.out.println("Gambar tidak dapat dikompresi.
Silakan coba lagi.");
        continue;
    }
    IOHandler.saveImage(compressedImage, absolutePath,
extension);

    // Output GIF hasil (jika diinginkan)
    if (saveGIF) {
        BufferedImage[] frames =
quadtree.GetFrames(extension);

```

```

        IOHandler.createGIF(frames, outputPathGIF, GIFname);
    }

    // Output waktu eksekusi
    System.out.println("Waktu kompresi: " + runTime + " ms");

    // Output ukuran gambar sebelum
    System.out.println("Ukuran file sebelum kompresi: " +
fileSizeBefore + " bytes");

    // Output ukuran gambar setelah
    long fileSizeAfter = IOHandler.getFileSize(absolutePath);
    System.out.println("Ukuran file setelah kompresi: " +
fileSizeAfter + " bytes");

    // Output persentase kompresi
    double compressionRatio =
Quadtree.compressionPercentage(fileSizeBefore, fileSizeAfter);
    System.out.printf("Persentase kompresi: %.2f%%\n",
compressionRatio);

    // Output kedalaman pohon
    System.out.println("Kedalaman pohon: " +
quadtree.getDepth());

    // Output banyak simpul pohon
    System.out.println("Jumlah simpul pohon: " +
quadtree.getNodeCount());

    System.out.println("-----
-----");
    }
}
}

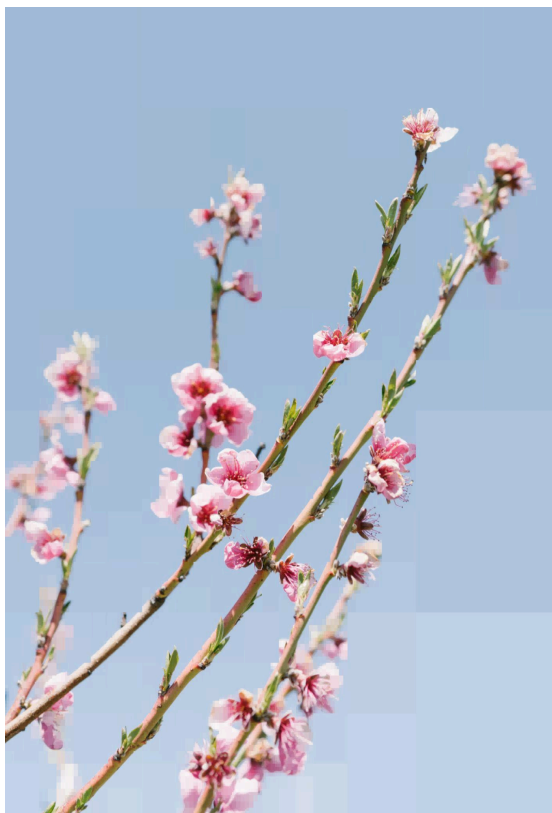
```

BAB IV

PERCOBAAN DAN ANALISIS


4.1. Percobaan

4.1.1. Test Case 1: Metode Error Variansi

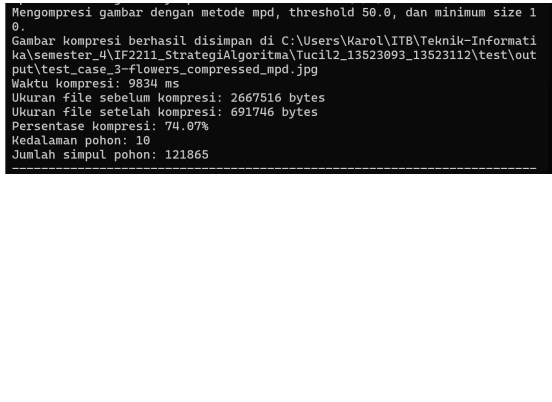
Masukan	Keluaran
<pre>PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor ----- Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 1 Silakan masukkan ambang batas: 100 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_1-flowers_compressed_variance" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n</pre> <p>Teks input: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" 1 100 10 0 "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\ "test_case_1-flowers_compressed_variance" n</p>	<pre>Mengompresi gambar dengan metode variance, threshold 100.0, dan minimum size 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_1-flowers_compressed_variance.jpg Waktu kompresi: 12620 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 684407 bytes Persentase kompresi: 74.34% Kedalaman pohon: 10 Jumlah simpul pohon: 138885</pre> 

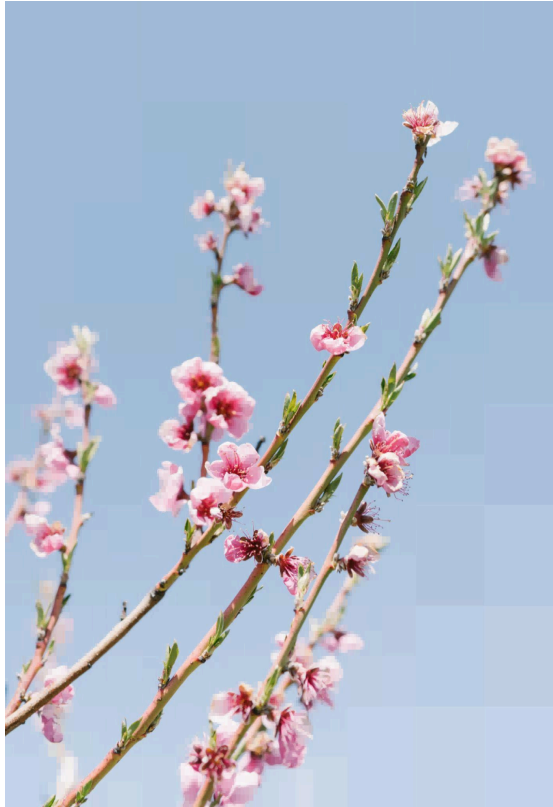
4.1.2. Test Case 2: Metode Error Mean Absolute Deviation

Masukan	Keluaran
---------	----------

<pre> PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor ----- Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 2 Silakan masukkan ambang batas: 10 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_2-flowers_compressed_mad" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n </pre> <p>Teks input:</p> <p>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg"</p> <p>2</p> <p>10</p> <p>10</p> <p>0</p> <p>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\"</p> <p>"test_case_2-flowers_compressed_mad"</p> <p>n</p>	<pre> Mengompresi gambar dengan metode mad, threshold 10.0, dan minimum size 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_2-flowers_compressed_mad.jpg Waktu kompresi: 11091 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 621311 bytes Persentase kompresi: 76.71% Kedalaman pohon: 10 Jumlah simpul pohon: 92653 </pre> 
--	---


4.1.3. Test Case 3: Metode Error *Maximum Pixel Difference*

Masukan	Keluaran
<pre> PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor ----- Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 3 Silakan masukkan ambang batas: 50 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_3-flowers_compressed_mpd" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n </pre> <p>Teks input:</p> <p>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg"</p>	<pre> Mengompresi gambar dengan metode mpd, threshold 50.0, dan minimum size 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_3-flowers_compressed_mpd.jpg Waktu kompresi: 9834 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 691746 bytes Persentase kompresi: 74.07% Kedalaman pohon: 10 Jumlah simpul pohon: 121865 </pre> 

rs.jpg" 3 50 10 0 "C:\Users\Karol\ITB\Teknik-Informatika\se mester_4\IF2211_StrategiAlgoritma\Tuci l2_13523093_13523112\test\output\ "test_case_3-flowers_compressed_mpd" n	
--	---


4.1.4. Test Case 4: Metode Error Entropi

Masukan	Keluaran
<pre>PS C:\Users\Karol\ITB\Teknik-Informatika\se mester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor ----- Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users \Karol\ITB\Teknik-Informatika\se mester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowe rs.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 4 Silakan masukkan ambang batas: 5 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB \Teknik-Informatika\se mester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\ Silakan masukkan nama file (tanpa ekstensi): "test_case_4-flowers_compre ssed_entropy" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n</pre> <p>Teks input: "C:\Users\Karol\ITB\Teknik-Informatika\se mester_4\IF2211_StrategiAlgoritma\Tuci l2_13523093_13523112\test\sample\flowe rs.jpg" 4 5 10 0</p>	<pre>Mengompresi gambar dengan metode entropy, threshold 5.0, dan minimum siz e 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informati ka\se mester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\out put\test_case_4-flowers_compressed_entropy.jpg Waktu kompresi: 7624 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 634536 bytes Persentase kompresi: 76.21% Kedalaman pohon: 10 Jumlah simpul pohon: 81449 -----</pre>

<pre>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" "test_case_4-flowers_compressed_entropy" " n</pre>	
---	---

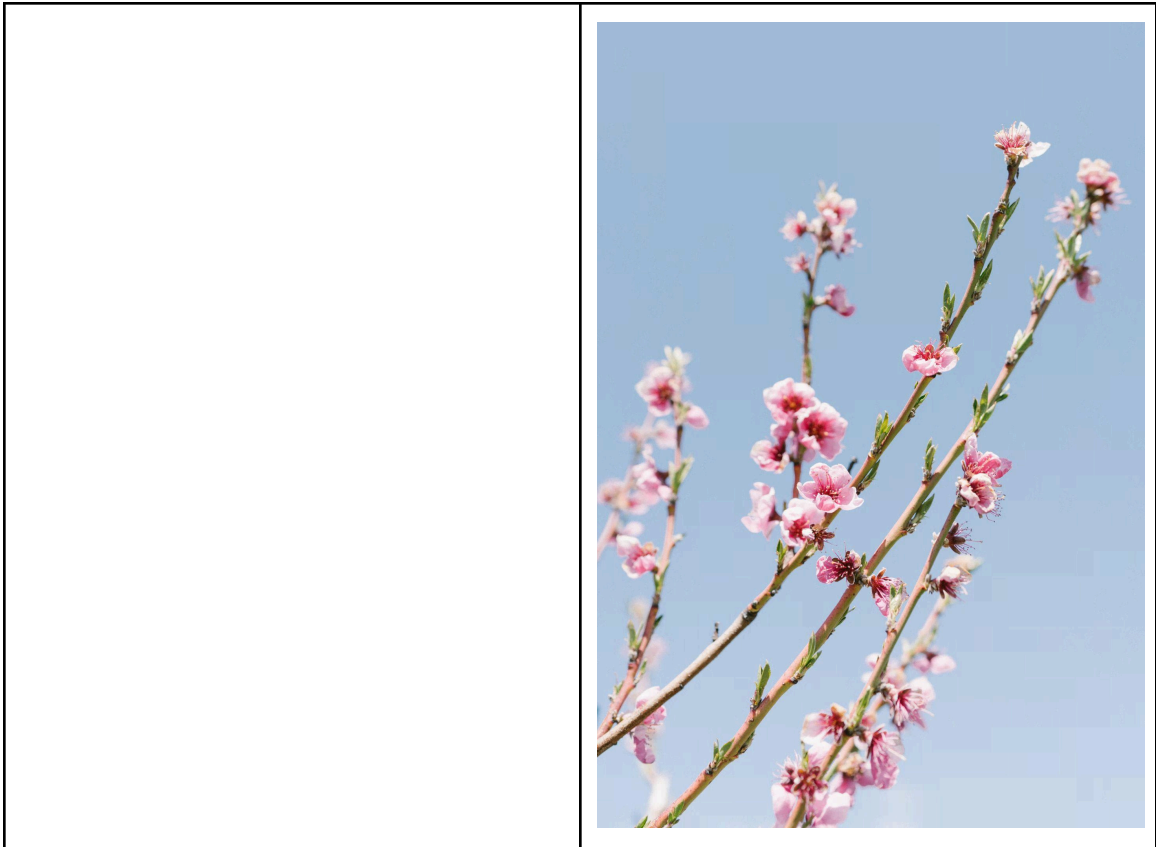
4.1.5. Test Case 5: Gambar dengan Kanal Alpha (Transparansi)

Masukan	Keluaran
<pre>PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\mario.png" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 1 Silakan masukkan ambang batas: 100 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_5-mario_compressed_alpha" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n</pre> <p>Teks input: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\mario.png" 1 100 10 0</p>	<pre>Mengompresi gambar dengan metode variance, threshold 100.0, dan minimum size 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_5-mario_compressed_alpha.png Waktu kompresi: 1556 ms Ukuran file sebelum kompresi: 513590 bytes Ukuran file setelah kompresi: 154270 bytes Persentase kompresi: 69.96% Kedalaman pohon: 8 Jumlah simpul pohon: 17357</pre>

<pre>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" "test_case_5-mario_compressed_alpha" n</pre>	
---	--

4.1.6. Test Case 6: Target Kompresi (Bonus)

Masukan	Keluaran
<pre>PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main ----- QuadTree Image Compressor ----- Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 1 Silakan masukkan ambang batas: 100 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0.7 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_6-flower_compressed_targeted" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): n Anda memilih untuk mengompresi gambar dengan target persentase 70.0% menggunakan metode variance dan minimum size 10. Silakan pilih batas atas pencarian threshold. Tekan enter untuk default (16256.25):</pre> <p>Teks input:</p> <pre>"C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" 1 100 10 0.7 "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" "test_case_6-flower_compressed_targeted" n [ENTER]</pre>	<pre>Mengompresi gambar dengan target persentase 70.0%, metode variance, threshold 100.0, minimum size 10, dan batas atas pencarian threshold 16256.25. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_6-flower_compressed_targeted.jpg Waktu kompresi: 49785 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 798188 bytes Persentase kompresi: 70.08% Kedalaman pohon: 12 Jumlah simpul pohon: 16029013</pre>



4.1.7. Test Case 7: GIF (Bonus)

Masukan	Keluaran
<pre>PS C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112> java -cp bin Main QuadTree Image Compressor Silakan masukkan alamat gambar, gunakan "/" sebagai seperator: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" Silakan pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Pilihan: 1 Silakan masukkan ambang batas: 100 Silakan masukkan ukuran blok minimum: 10 Silakan masukkan persentase kompresi (0..1): 0 Silakan masukkan alamat absolut output (tanpa nama): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_7-flower_compressed.gif" Apakah Anda ingin menyimpan dalam bentuk GIF? (Y/N): y Silakan masukkan alamat GIF absolut (tanpa nama file): "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\" Silakan masukkan nama file (tanpa ekstensi): "test_case_7-flower_compressed.gif"</pre> <p>Teks input: "C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\sample\flowers.jpg" 1 100</p>	<pre>Mengompresi gambar dengan metode variance, threshold 100.0, dan minimum size 10. Gambar kompresi berhasil disimpan di C:\Users\Karol\ITB\Teknik-Informatika\semester_4\IF2211_StrategiAlgoritma\Tucil2_13523093_13523112\test\output\test_case_7-flower_compressed.gif.jpg Waktu kompresi: 13062 ms Ukuran file sebelum kompresi: 2667516 bytes Ukuran file setelah kompresi: 684407 bytes Persentase kompresi: 74.34% Kedalaman pohon: 10 Jumlah simpul pohon: 138885</pre>

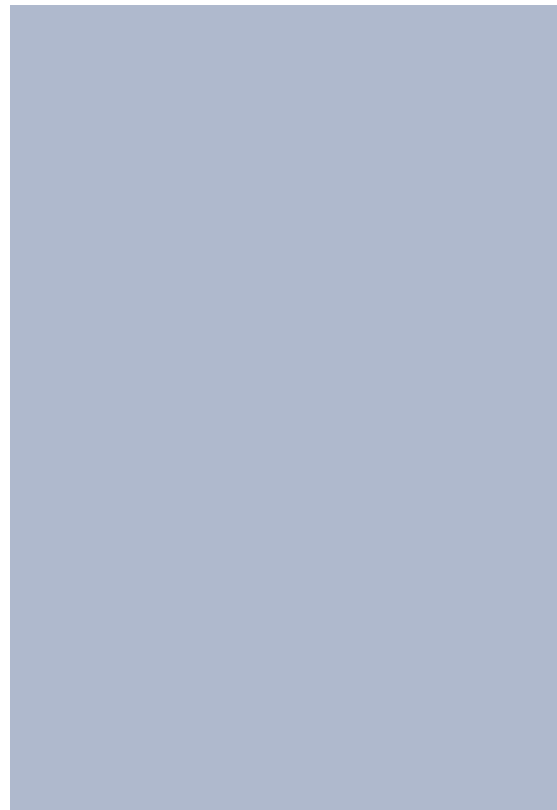
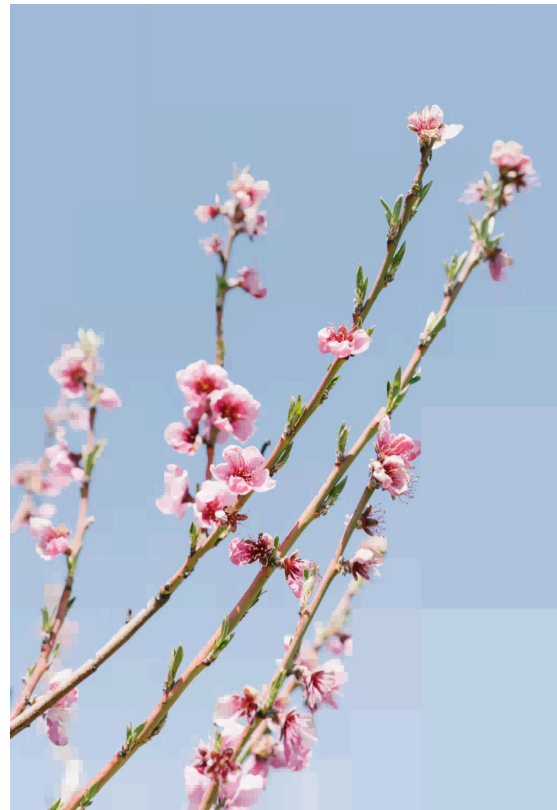
10

0

"C:\Users\Karol\ITB\Teknik-Informatika\s
emester_4\IF2211_StrategiAlgoritma\Tuci
l2_13523093_13523112\test\output\
"test_case_7-flower_compressed_gif"

y

"C:\Users\Karol\ITB\Teknik-Informatika\s
emester_4\IF2211_StrategiAlgoritma\Tuci
l2_13523093_13523112\test\output\
"test_case_7-flower_compressed_gif"



(File GIF dapat diunduh dari repository)

4.2. Analisis

4.2.1. Kompleksitas Algoritma

Dalam program ini, algoritma *divide and conquer* diterapkan saat membangun Quadtree dari gambar yang dimasukkan. Kompleksitas algoritma keseluruhan dianalisis dengan mempertimbangkan kompleksitas dari kalkulasi error dan pembuatan Node anak. Kalkulasi error dengan metode variansi, *mean absolute deviation*, *max pixel difference*, dan entropi memiliki kompleksitas $O(n)$ dengan n adalah jumlah pixel pada blok yang diproses. Jumlah proses pada algoritma *divide and conquer* dapat dinyatakan secara rekursif seperti pada (1).

$$T(n) = 4T\left(\frac{n}{4}\right) + O(n) \quad (1)$$

Pada (1), hal yang dapat diperhatikan adalah sebagai berikut:

- Koefisien 4 menyatakan jumlah Node anak yang dimiliki Node saat ini.
- $T\left(\frac{n}{4}\right)$ menyatakan jumlah proses yang dijalankan saat memeriksa Node anak yang memiliki jumlah pixel blok $\frac{1}{4}$ dari jumlah pixel pada blok saat ini.
- $O(n)$ adalah kompleksitas dari proses kalkulasi error yang dijalankan pada setiap Node.

Menggunakan Teorema Master, kompleksitas algoritma *divide and conquer* pada program berdasarkan definisi rekursif tersebut dapat ditentukan yakni (2).

$$T(n) = O(n \log(n)) \quad (2)$$

4.2.2. Analisis Percobaan

Percobaan dilakukan menggunakan 2 jenis gambar, yakni flowers.jpg sebagai gambar utama dan mario.png untuk kasus kompresi gambar yang memiliki kanal alpha. Secara umum, program memberikan hasil yang sesuai ekspektasi. Gambar yang dihasilkan memiliki pola kotak-kotak pada bagian dengan warna dan tekstur yang konsisten, menandakan algoritma dan perhitungan error berhasil mendeteksi blok pada gambar yang tidak memiliki fitur mencolok lalu mengompresnya menjadi satu warna. Di saat yang sama, bagian pada gambar dengan fitur yang mencolok seperti bagian bunga yang berwarna kontras masih

dipertahankan ketajamannya karena algoritma menghasilkan upa pohon yang lebih banyak untuk merepresentasikan bagian tersebut.

Pada setiap *test case* percobaan, digunakan nilai ambang batas yang berbeda-beda. Ambang batas maksimum setiap jenis galat didapatkan dengan metode yang berbeda pula. Ambang batas maksimum variansi dan *mean absolute deviation* didapat dengan memaksimumkan persamaannya dengan berdasarkan nilai maksimum pixel 8-bit yakni 255. Dengan metode tersebut, didapat batas atas variansi dan *mean absolute deviation* adalah 16256,25 dan 127,5 secara berturut-turut. Ambang batas galat *max pixel difference* adalah selisih dari nilai terbesar dan terkecil dari sebuah pixel 8-bit yakni 255. Galat berbasis entropi memiliki cara lain untuk membuktikan nilai maksimumnya yakni menggunakan Ketaksamaan Jensen. Menggunakan ketaksamaan tersebut, nilai maksimum entropi untuk data pixel 8-bit adalah 8.

Metode variansi menggunakan nilai 100 karena *range* variansi. Metode *mean absolute deviation* memiliki batas atas yang lebih kecil sehingga threshold ditetapkan lebih rendah yakni 10 agar hasil kompresi masih memiliki tekstur yang jelas. Namun, hasil percobaan menunjukkan bahwa ambang batas tersebut masih menghasilkan gambar dengan kualitas yang lebih rendah daripada *test case 1* sehingga ambang batas masih bisa diturunkan lagi. Percobaan dengan metode *max pixel difference* dan entropi juga menggunakan nilai ambang batas yang proporsional dengan *range*-nya untuk mendapatkan hasil yang masih mempertahankan fitur mencolok gambar awal. Penanganan khusus diterapkan pada gambar dengan kanal alpha seperti gambar PNG pada *test case 5*. Dalam kasus ini, galat tidak dihitung untuk kanal alpha dan nilai pada kanal tersebut langsung dipindahkan dengan nilai yang sama ke gambar akhir untuk mempertahankan fitur transparansi gambar awal.

Skema target kompresi diprogram dengan strategi pencarian biner. Pertama-tama, gambar dikompresi dengan membangun pohon Quadtree yang memiliki nilai ambang batas berupa nilai tengah *range* untuk metode perhitungan galat yang dipilih. Pada setiap iterasi pencarian, pohon yang telah dibangun tidak dibangun ulang, melainkan Node ditandai sebagai daun semu apabila memenuhi

kriteria sebagai daun untuk ambang batas yang baru. Metode ini meningkatkan efisiensi pencarian dibandingkan versi awal program yang membangun ulang pohon pada setiap iterasi.

Fitur program GIF dibuat memanfaatkan beberapa modul bawaan dari package `javax.imageio` untuk menghasilkan *sequence* GIF dari sebuah array `BufferedImage`. Setiap *frame* dalam GIF adalah gambar yang dihasilkan dari Quadtree yang telah dibangun namun dibangun kembali dengan kedalaman baru dan berbeda secara berurutan untuk setiap frame yang berurutan. Metode ini menciptakan animasi seakan-akan gambar sedang dibangun dari Node root hingga mencapai kedalaman tertentu.

LAMPIRAN

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	✓	

REFERENSI

Munir, R. (2025). Algoritma Divide and Conquer (Bagian 1) [Materi kuliah]. Institut Teknologi Bandung. Diakses dari [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)