LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

KOMPRESI GAMBAR DENGAN METODE QUADTREE

Disusun Oleh:

Jonathan Kenan Budianto 13523139

Indah Novita Tangdililing 13523047

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG, 40132

2025

DAFTAR ISI

BAB I
LANDASAN TEORI

1.1. Pengolahan Citra Digital

Pengolahan citra digital adalah cabang ilmu komputer yang berkaitan dengan manipulasi dan analisis gambar dalam bentuk digital. Gambar digital terdiri atas kumpulan titik-titik kecil yang disebut piksel (pixel), yang masing-masing mewakili satu titik warna. Dalam sistem warna RGB (Red, Green, Blue), setiap piksel memiliki tiga nilai yang menunjukkan intensitas warna merah, hijau, dan biru.

Nilai-nilai ini biasanya berada dalam rentang 0 hingga 255. Kombinasi dari ketiga warna tersebut akan menentukan warna akhir dari piksel. Karena sebuah gambar bisa terdiri dari jutaan piksel, maka ukuran file gambar bisa menjadi sangat besar. Oleh karena itu, pengolahan citra digital sering digunakan untuk memperkecil ukuran gambar atau menyesuaikan kualitasnya, baik untuk keperluan penyimpanan, transmisi, atau analisis.

1.2. Kompresi Gambar

Kompresi gambar adalah proses mengurangi ukuran file gambar agar lebih efisien disimpan atau dikirimkan. Dalam kehidupan sehari-hari, kompresi gambar digunakan ketika seseorang mengunggah foto ke media sosial, mengirim gambar melalui pesan, atau menyimpan banyak gambar dalam perangkat dengan keterbatasan ruang penyimpanan.

Terdapat dua jenis utama kompresi gambar:

Lossless Compression (kompresi tanpa kehilangan): Teknik ini mempertahankan semua data gambar asli. Artinya, gambar yang telah dikompresi bisa dikembalikan ke bentuk aslinya tanpa kehilangan informasi apa pun. Contohnya adalah format PNG.Lossy Compression (kompresi dengan kehilangan): Teknik ini menghapus beberapa informasi yang dianggap tidak terlalu penting dari gambar, sehingga ukuran file bisa jauh lebih kecil. Namun, proses ini tidak bisa dikembalikan sepenuhnya ke gambar aslinya. Contohnya adalah format JPEG.

Dalam tugas ini, kompresi dilakukan dengan pendekatan lossy, yaitu mengurangi ukuran file gambar dengan mengorbankan sebagian kecil informasi visual yang dianggap tidak terlalu signifikan.

1.3. Divide and Conquer

Divide and Conquer (pecah dan taklukkan) adalah salah satu strategi dasar dalam desain algoritma. Ide dasarnya adalah memecah suatu masalah besar menjadi beberapa sub-masalah yang lebih kecil, menyelesaikan sub-masalah tersebut secara rekursif, dan menggabungkan hasilnya untuk membentuk solusi akhir.

Tiga langkah utama dalam divide and conquer:

- Divide: Memecah masalah menjadi sub-masalah yang lebih kecil.
- Conquer: Menyelesaikan setiap sub-masalah. Jika sub-masalahnya cukup kecil, langsung diselesaikan; jika tidak, dipecah lagi.
- Combine: Menggabungkan hasil dari sub-masalah untuk membentuk solusi akhir.

Metode ini digunakan dalam banyak algoritma terkenal seperti Merge Sort, Quick Sort, dan juga dalam kompresi gambar berbasis Quadtree yang digunakan dalam tugas ini.

1.4. Quadtree

Quadtree adalah struktur data pohon yang setiap simpulnya memiliki paling banyak empat anak. Struktur ini sangat berguna untuk merepresentasikan data dua dimensi, seperti gambar, karena ia mampu membagi bidang dua dimensi menjadi empat bagian (kuadran) secara rekursif.

Dalam pengolahan citra, Quadtree digunakan untuk membagi gambar menjadi blok-blok kecil. Prosesnya dimulai dari seluruh gambar, kemudian diperiksa apakah warna dalam blok tersebut cukup seragam. Jika seragam, maka blok tersebut menjadi simpul daun dan tidak dibagi lebih lanjut. Jika tidak seragam, maka blok dibagi menjadi empat kuadran dan proses ini diulangi untuk masing-masing kuadran.

Setiap simpul dalam struktur Quadtree biasanya menyimpan informasi posisi (x, y), ukuran (width, height), dan rata-rata warna blok. Representasi ini membuat pengolahan data gambar menjadi lebih efisien, karena tidak perlu menyimpan informasi piksel satu per satu, cukup dengan menyimpan simpul yang mewakili blok seragam.

1.5. Kompresi Gambar dengan Quadtree

Kompresi gambar dengan metode Quadtree bekerja berdasarkan prinsip divide and conquer. Gambar diolah sebagai matriks piksel warna, dan Quadtree digunakan untuk membagi gambar ke dalam blok-blok kecil berdasarkan tingkat keseragaman warna.

Setiap blok diuji menggunakan metode perhitungan error (seperti variansi atau deviasi), untuk menentukan apakah blok tersebut cukup seragam. Jika tidak, maka blok tersebut dibagi menjadi empat bagian dan proses dilakukan secara rekursif.

Jika blok sudah cukup seragam, maka tidak perlu dibagi lagi, dan nilai warnanya bisa dinormalisasi menjadi nilai rata-rata warna blok tersebut. Dengan cara ini, gambar dapat disimpan hanya dengan informasi blok-blok seragam, yang secara signifikan mengurangi ukuran file dibandingkan dengan menyimpan setiap piksel satu per satu.

Metode ini sangat efisien terutama untuk gambar yang memiliki banyak area seragam, seperti sketsa, peta, atau gambar animasi sederhana. Selain itu, kompresi dengan Quadtree juga cocok digunakan sebagai pendekatan visual yang tetap mempertahankan bentuk umum gambar meskipun ada penghilangan detail kecil.

BAB II
ALGORITMA DIVIDE AND CONQUER

2.1. Langkah-langkah Algoritma Divide and Conquer

1. Divide: Membagi blok jadi 4 bagian.
2. Conquer: Proses rekursif pada masing-masing sub-blok.
3. Combine: Bangun node dari anak-anaknya atau simpul akhir (leaf).

2.1.1. Ambil sebuah blok citra (bagian gambar)

Mulai dari seluruh gambar, dianggap sebagai satu blok persegi panjang (atau persegi) berukuran penuh. mengambil sebuah blok citra, yaitu bagian dari gambar yang akan dianalisis. Pada awalnya, blok ini merupakan keseluruhan gambar yang dianggap sebagai satu kesatuan area berbentuk persegi atau persegi panjang. Blok citra ini terdiri dari kumpulan piksel yang merepresentasikan warna atau intensitas cahaya pada setiap titik dalam area tersebut. Proses ini berfungsi sebagai titik awal sebelum dilakukan pengecekan lebih lanjut terhadap karakteristik blok tersebut, seperti homogenitas warnanya. Dengan memulai dari satu blok besar, algoritma kemudian dapat menentukan apakah blok tersebut cukup seragam atau perlu dibagi lebih lanjut menjadi bagian-bagian yang lebih kecil. Pendekatan ini memungkinkan analisis gambar dilakukan secara hierarkis dan bertahap, dimulai dari gambaran besar dan kemudian menyusuri ke detail-detail terkecil bila diperlukan.

2.1.2. Evaluasi homogenitas blok

Periksa apakah blok tersebut cukup homogen berdasarkan 2 kriteria yaitu yang pertama pengecekan nilai error sesuai dengan **ambas batas (threshold)** (yaitu variance, Mean Absolute Deviation (MAD), Max Pixel Difference, Entropy , atau SSIM sesuai dengan metode yang di input). Blok masih bersifat homogen saat nilai error suatu blok **berada dibawah** threshold. Kriteria blok sudah cukup homogen yang kedua adalah

ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah **ukuran blok nilai minimum.**

Jika blok cukup homogen, maka:

- Tidak perlu dibagi lagi.
- Blok disimpan sebagai leaf node dalam quadtree dengan warna rata-rata dari blok tersebut.

2.1.3. Jika tidak homogen, bagi menjadi 4 kuadran

Jika sebuah blok citra tidak memenuhi syarat homogenitas artinya blok tersebut memiliki variasi warna atau intensitas yang terlalu tinggi berdasarkan ambang batas tertentu. **Maka langkah selanjutnya dalam algoritma Quadtree adalah membagi blok tersebut menjadi empat kuadran.** Pembagian ini dilakukan secara merata, menghasilkan empat sub-blok yang lebih kecil: kiri atas, kanan atas , kiri bawah , dan kanan bawah . Setiap sub-blok ini mewakili seperempat bagian dari blok asal dan akan diproses secara terpisah. Tujuan dari pembagian ini adalah untuk mengisolasi bagian-bagian yang lebih kecil dari gambar sehingga dapat dianalisis dengan lebih mendetail. Dengan membagi blok yang tidak homogen, algoritma berusaha menemukan area-area kecil yang cukup seragam sehingga tidak perlu diproses lebih lanjut. Pendekatan ini juga mencerminkan prinsip.

2.1.4. Rekursi pada tiap sub-blok

Lakukan langkah 2 dan 3 secara rekursif pada tiap sub-blok. Proses pembagian akan terus berlanjut hingga kondisi homogenitas terpenuhi, atau ukuran minimum blok telah tercapai. Setelah sebuah blok citra dibagi menjadi empat sub-blok karena tidak memenuhi syarat homogenitas, langkah selanjutnya dalam algoritma Quadtree adalah melakukan rekursi pada tiap sub-blok tersebut. Artinya, keempat kuadran (kiri atas, kanan atas, kiri bawah, dan kanan bawah) akan diproses kembali menggunakan langkah-langkah yang sama seperti pada blok induknya. Setiap sub-blok akan diperiksa apakah ia cukup homogen berdasarkan kriteria tertentu. Jika ya, maka ia akan menjadi simpul daun dalam pohon quadtree. Namun, jika tidak, maka ia

akan kembali dibagi menjadi empat sub-blok yang lebih kecil, dan proses rekursi akan berlanjut. Pendekatan rekursif ini memungkinkan algoritma untuk secara otomatis menyusuri dan menguraikan gambar hanya pada bagian-bagian yang tidak seragam, sambil mempertahankan bagian-bagian yang cukup seragam untuk tetap utuh. Dengan cara ini, algoritma dapat menghasilkan representasi gambar yang efisien, baik dalam hal struktur data maupun pemrosesan.

2.1.5. Gabungkan hasil rekursi

Setelah seluruh proses rekursi pada sub-blok selesai, informasi dari masing-masing hasil rekursi ini dikumpulkan dan disusun ke dalam satu node induk. Node ini akan menyimpan posisi koordinat blok asli, ukuran blok, nilai warna rata-rata (jika diperlukan), dan referensi ke keempat node anaknya. Inilah yang dimaksud dengan penggabungan hasil rekursi — membentuk kembali satu simpul pohon dari gabungan hasil pemrosesan keempat sub-blok. Proses ini berlangsung terus-menerus ke atas hingga mencapai root (simpul awal), sehingga membentuk struktur pohon penuh dari bawah ke atas. Tahap ini penting karena tanpanya, kita tidak dapat membangun Quadtree secara utuh dan terstruktur, dan tidak akan bisa merekonstruksi citra secara efisien dari pohon tersebut.

2.2. Pseudocode Algoritma Divide and Conquer

2.2.1 Divide and Conquer

```
Function buildQuadtree(pixels, x, y, width, height):
  avgColor ← GetAverageColor(pixels, x, y, width, height)
  If ShouldSplit(pixels, x, y, width, height, avgColor, method, threshold) is True:
       node ← CreateNewNode(x, y, width, height, avgColor)

       halfWidth ← width / 2
       halfHeight ← height / 2

       node.children[0] ← buildQuadtree(pixels, x, y, halfWidth, halfHeight)
                // Top-left quadrant
       node.children[1] ← buildQuadtree(pixels, x + halfWidth, y, width - halfWidth,
       halfHeight)
                // Top-right quadrant
       node.children[2] ← buildQuadtree(pixels, x, y + halfHeight, halfWidth, height -
       halfHeight)
                // Bottom-left quadrant
       node.children[3] ← buildQuadtree(pixels, x + halfWidth, y + halfHeight, width -
       halfWidth, height - halfHeight)
                // Bottom-right quadrant
    Return node
  Else:
    Return CreateLeafNode(x, y, width, height, avgColor)
```

2.2.1 Combine

```
Function reconstructImageByDepth(node, image, originalWidth, originalHeight,
currentDepth):

    If node is null:
        Return

    If currentDepth == 1 OR node has no children:
        For i from 0 to node.height - 1:
            For j from 0 to node.width - 1:
                If (node.y + i) < originalHeight AND (node.x + j) < originalWidth:
                    image[node.y + i][node.x + j] ← node.avgColor
        Return

    hasValidChildren ← False

    For each child in node.children:
        If child is not null:
            hasValidChildren ← True
            Call reconstructImageByDepth(child, image, originalWidth,
originalHeight, currentDepth - 1)

    If hasValidChildren is False:
        For i from 0 to node.height - 1:
            For j from 0 to node.width - 1:
                If (node.y + i) < originalHeight AND (node.x + j) < originalWidth:
                    image[node.y + i][node.x + j] ← node.avgColor
```

BAB III

SOURCE PROGRAM

3.1. Struktur folder dan file

```
src/
├── Main.java
├── compressor/
│   └── Calculate.java
├── image/
│   ├── GifSequenceWriter.java
│   └── ImageUtils.java
└── quadtree/
    ├── QuadtreeBuilder.java
    └── QuadtreeNode.java
```

3.2. Source Code

3.2.1 Main.java

```java
import java.util.*;
import java.io.*;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import javax.imageio.stream.FileImageOutputStream;

import image.ImageUtils;
import quadtree.QuadtreeBuilder;
import quadtree.QuadtreeNode;
import image.GifSequenceWriter;



public class Main {
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);
```

```
        System.out.print("Masukkan alamat absolut gambar yang akan dikompresi: ");
        String inputImagePath = scanner.nextLine();

        System.out.println("Pilih metode perhitungan error (1-5):");
        System.out.println("1. Variance");
        System.out.println("2. Mean Absolute Deviation (MAD)");
        System.out.println("3. Max Pixel Difference");
        System.out.println("4. Entropy");
        System.out.print("Masukkan nomor metode: ");
        int method = scanner.nextInt();

        System.out.print("Masukkan ambang batas: ");
        double threshold = scanner.nextDouble();

        System.out.print("Masukkan ukuran blok minimum: ");
        int minSize = scanner.nextInt();

        scanner.nextLine();

        System.out.print("Masukkan alamat absolut gambar hasil kompresi (termasuk
nama file dan ekstensi): ");
        String outputImagePath = scanner.nextLine();

        System.out.print("Masukkan alamat absolut GIF (bonus, kosongkan jika tidak
ada): ");
        String gifOutputPath = scanner.nextLine();

        try {
            File inputFile = new File(inputImagePath);
            if (!inputFile.exists()) {
                System.err.println("File gambar tidak ditemukan: " + inputImagePath);
                return;
            }

            BufferedImage originalImage = null;
            try {
                originalImage = ImageIO.read(inputFile);
            } catch (IOException e) {
                System.err.println("Gagal membaca file gambar (IOException): " +
inputImagePath + " - " + e.getMessage());
                return;
```

```
            }

            if (originalImage == null) {
                System.err.println("Gagal membaca file gambar (Format tidak didukung
atau rusak): " + inputImagePath);
                return;
            }
            try {
                BufferedImage testImage = ImageIO.read(new File(inputImagePath));
                System.out.println("Gambar berhasil dibaca menggunakan program
Java.");
            } catch (IOException e) {
                System.err.println("Gagal membaca gambar menggunakan program Java: "
+ e.getMessage());
            }

            int[][][] pixelData = ImageUtils.loadImage(inputImagePath);
            int height = pixelData.length;
            int width = pixelData[0].length;

            long startTime = System.nanoTime();
            long executionTime;

            if (pixelData.length > 0 && pixelData[0].length > 0) {
                QuadtreeNode root = QuadtreeBuilder.buildQuadtree(pixelData, 0, 0,
width, height, threshold, minSize, method, pixelData);
                int[][][] compressedPixelData = new int[height][width][3];
                QuadtreeBuilder.reconstructImageByDepth(root, compressedPixelData,
width, height, QuadtreeBuilder.calculateTreeDepth(root));

                ImageUtils.saveImage(compressedPixelData, outputImagePath, "jpg");
                File originalFile = new File(inputImagePath);
                File compressedFile = new File(outputImagePath);
                int originalSize = (int) originalFile.length();
                int compressedSize = (int) compressedFile.length();
                double compressionPercentage =
ImageUtils.calculateCompressionPercentage(originalSize, compressedSize);

                executionTime = (System.nanoTime() - startTime) / 1000000;
//milliseconds
```

```
                System.out.println("Waktu eksekusi: " + executionTime + " ms");
                System.out.println("Ukuran gambar sebelum: " + originalSize + "
bytes");
                System.out.println("Ukuran gambar setelah: " + compressedSize + "
bytes");
                System.out.println("Persentase kompresi: " + String.format("%.2f",
compressionPercentage) + " %");

                int treeDepth = QuadtreeBuilder.calculateTreeDepth(root);
                int nodeCount = QuadtreeBuilder.calculateNodeCount(root);
                System.out.println("Kedalaman pohon: " + treeDepth);
                System.out.println("Banyak simpul pada pohon: " + nodeCount);

                if (!gifOutputPath.isEmpty()) {
                    List<BufferedImage> gifImages = new ArrayList<>();

                    for (int depth = 1; depth <= treeDepth; depth++) {
                        // Rekonstruksi gambar berdasarkan kedalaman tertentu
                        int[][][] depthCompressedPixelData = new
int[height][width][3];
                        QuadtreeBuilder.reconstructImageByDepth(root,
depthCompressedPixelData, width, height, depth);

                        // Konversi data piksel ke BufferedImage
                        BufferedImage frame =
QuadtreeBuilder.convertToBufferedImage(depthCompressedPixelData);
                        gifImages.add(frame);
                    }

                    GifSequenceWriter gifWriter = new GifSequenceWriter(
                        new FileImageOutputStream(new File(gifOutputPath)),
                        BufferedImage.TYPE_INT_RGB,
                        500, // Delay antara frame dalam milidetik
                        true // Loop terus-menerus
                    );

                    for (BufferedImage frame : gifImages) {
                        gifWriter.writeToSequence(frame);
                    }
                    gifWriter.close();
                    System.out.println("GIF berhasil dibuat: " + gifOutputPath);
```

```
                } else {
                    System.out.println("GIF tidak dibuat.");
                }
            } else {
                System.err.println("Gambar kosong atau tidak valid.");
            }


        } catch (IOException e) {
            System.err.println("Terjadi kesalahan: " + e.getMessage());
        } catch (Exception e) {
            System.err.println("Terjadi kesalahan tak terduga: " + e.getMessage());
        } finally {
            scanner.close();
        }
    }
}
```

3.2.2 Calculate.java

```
package compressor;

public class Calculate{

    public static int[] calculateAverageColor(int[][][] block) {
        int N = block.length * block[0].length;
        int[] sum = {0, 0, 0};

        for (int[][] row : block) {
            for (int[] pixel : row) {
                sum[0] += pixel[0];
                sum[1] += pixel[1];
                sum[2] += pixel[2];
            }
        }
        return new int[]{sum[0] / N, sum[1] / N, sum[2] / N};
    }
```

```
    public static double[] calculateVariance(int[][][] block, int[] avgColor) {
        int N = block.length * block[0].length;
        double[] variance = {0.0, 0.0, 0.0};

        for (int[][] row : block) {
            for (int[] pixel : row) {
                for (int c = 0; c < 3; c++) {
                    variance[c] += Math.pow(pixel[c] - avgColor[c], 2);
                }
            }
        }
        variance[0] /= N;
        variance[1] /= N;
        variance[2] /= N;

        double varianceRGB = (variance[0] + variance[1] + variance[2]) / 3.0;
        return new double[]{variance[0], variance[1], variance[2], varianceRGB};
    }

    public static double[] calculateMAD(int[][][] block, int[] avgColor) {
        int N = block.length * block[0].length;
        double[] mad = {0.0, 0.0, 0.0};

        for (int[][] row : block) {
            for (int[] pixel : row) {
                for (int c = 0; c < 3; c++) {
                    mad[c] += Math.abs(pixel[c] - avgColor[c]);
                }
            }
        }
        return new double[]{mad[0] / N, mad[1] / N, mad[2] / N, ((mad[0] + mad[1] +
mad[2])/ N) / 3.0};
    }

    public static double[] calculateMaxPixelDifference(int[][][] block) {
        int minRGB[] = {255, 255, 255};
        int maxRGB[] = {0, 0, 0};

        for (int[][] row : block) {
            for (int[] pixel : row) {
                for (int c = 0; c < 3; c++) {
```

```
                    minRGB[c] = Math.min(minRGB[c], pixel[c]);
                    maxRGB[c] = Math.max(maxRGB[c], pixel[c]);
                }
            }
        }
        return new double[]{
            maxRGB[0] - minRGB[0],
            maxRGB[1] - minRGB[1],
            maxRGB[2] - minRGB[2],
            (maxRGB[0] - minRGB[0] + maxRGB[1] - minRGB[1] + maxRGB[2] - minRGB[2]) /
3.0
        };
    }

    public static double[] calculateEntropy(int[][][] block) {
        double[] entropy = {0.0, 0.0, 0.0};
        int totalPixels = block.length * block[0].length;

        for (int c = 0; c < 3; c++) {
            int[] histogram = new int[256];
            for (int[][] row : block) {
                for (int[] pixel : row) {
                    histogram[pixel[c]]++;
                }
            }

            for (int i = 0; i < 256; i++) {
                if (histogram[i] > 0) {
                    double p = (double) histogram[i] / totalPixels;
                    entropy[c] -= p * (Math.log(p) / Math.log(2));
                }
            }
        }
        return new double[]{entropy[0], entropy[1], entropy[2], (entropy[0] +
entropy[1] + entropy[2]) / 3.0};
    }
```

3.2.3 GifSequenceWriter.java

```java
package image;

import javax.imageio.*;
import javax.imageio.metadata.*;
import javax.imageio.stream.*;
import java.awt.image.*;
import java.io.*;
import java.util.Iterator;


public class GifSequenceWriter {
   protected ImageWriter gifWriter;
   protected ImageWriteParam imageWriteParam;
   protected IIOMetadata imageMetaData;


   public GifSequenceWriter(
       ImageOutputStream outputStream,int imageType, int timeBetweenFramesMS,
       boolean loopContinuously) throws IIOException, IOException {
     gifWriter = getWriter();
     imageWriteParam = gifWriter.getDefaultWriteParam();
     ImageTypeSpecifier imageTypeSpecifier =
       ImageTypeSpecifier.createFromBufferedImageType(imageType);
      imageMetaData =
       gifWriter.getDefaultImageMetadata(imageTypeSpecifier,
       imageWriteParam);
      String metaFormatName = imageMetaData.getNativeMetadataFormatName();
      IIOMetadataNode root = (IIOMetadataNode)
       imageMetaData.getAsTree(metaFormatName);
      IIOMetadataNode graphicsControlExtensionNode = getNode(
       root,
       "GraphicControlExtension");
      graphicsControlExtensionNode.setAttribute("disposalMethod", "none");
     graphicsControlExtensionNode.setAttribute("userInputFlag", "FALSE");
     graphicsControlExtensionNode.setAttribute(
       "transparentColorFlag",
       "FALSE");
     graphicsControlExtensionNode.setAttribute(
       "delayTime",
       Integer.toString(timeBetweenFramesMS / 10));
     graphicsControlExtensionNode.setAttribute(
```

```
      "transparentColorIndex",
      "0");
    IIOMetadataNode commentsNode = getNode(root, "CommentExtensions");
   commentsNode.setAttribute("CommentExtension", "Created by MAH");
    IIOMetadataNode appEntensionsNode = getNode(
     root,
     "ApplicationExtensions");
    IIOMetadataNode child = new IIOMetadataNode("ApplicationExtension");
    child.setAttribute("applicationID", "NETSCAPE");
   child.setAttribute("authenticationCode", "2.0");
    int loop = loopContinuously ? 0 : 1;
    child.setUserObject(new byte[]{ 0x1, (byte) (loop & 0xFF), (byte)
     ((loop >> 8) & 0xFF)});
   appEntensionsNode.appendChild(child);
    imageMetaData.setFromTree(metaFormatName, root);
    gifWriter.setOutput(outputStream);
    gifWriter.prepareWriteSequence(null);
}

public void writeToSequence(RenderedImage img) throws IOException {
  gifWriter.writeToSequence(
    new IIOImage(
      img,
      null,
      imageMetaData),
    imageWriteParam);
}



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

private static ImageWriter getWriter() throws IIOException {
  Iterator<ImageWriter> iter = ImageIO.getImageWritersBySuffix("gif");
  if(!iter.hasNext()) {
    throw new IIOException("No GIF Image Writers Exist");
  } else {
    return iter.next();
  }
}
```

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

3.2.4 ImageUtils.java

```
package image;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class ImageUtils {

   public static int[][][] loadImage(String imagePath) throws IOException {
       File inputFile = new File(imagePath);
       if (!inputFile.exists()) {
           throw new IOException("File not found: " + imagePath);
       }

       BufferedImage originalImage = ImageIO.read(inputFile);
       if (originalImage == null) {
           throw new IOException("Failed to read image (Unsupported format or
corrupted file): " + imagePath);
```

```
        }

        int width = originalImage.getWidth();
        int height = originalImage.getHeight();
        int[][][] pixelData = new int[height][width][3];

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int rgb = originalImage.getRGB(x, y);
                pixelData[y][x][0] = (rgb >> 16) & 0xFF; // R
                pixelData[y][x][1] = (rgb >> 8) & 0xFF;  // G
                pixelData[y][x][2] = rgb & 0xFF;         // B
            }
        }

        return pixelData;
    }
    public static int[][][] extractBlock(int[][][] image, int x, int y, int width,
int height) {
        int imageHeight = image.length;
        int imageWidth = image[0].length;

        int actualWidth = Math.min(width, imageWidth - x);
        int actualHeight = Math.min(height, imageHeight - y);

        int[][][] block = new int[actualHeight][actualWidth][3];
        for (int i = 0; i < actualHeight; i++) {
            for (int j = 0; j < actualWidth; j++) {
                block[i][j] = image[y + i][x + j];
            }
        }
        return block;
    }
    public static void saveImage(int[][][] pixelData, String outputPath, String
format) throws IOException {
        int height = pixelData.length;
        int width = pixelData[0].length;

        BufferedImage compressedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        for (int y = 0; y < height; y++) {
```

```
            for (int x = 0; x < width; x++) {
                int rgb = (pixelData[y][x][0] << 16) | (pixelData[y][x][1] << 8) |
pixelData[y][x][2];
                compressedImage.setRGB(x, y, rgb);
            }
        }

        ImageIO.write(compressedImage, format, new File(outputPath));
    }

    public static double calculateCompressionPercentage(int originalSize, int
compressedSize) {
        return (1.0 - ((double) compressedSize / originalSize)) * 100.0;
    }
    public static int getPowerOfTwoSize(int size) {
        int power = 1;
        while (power < size) {
            power *= 2;
        }
        return power;
    }

}
```

3.2.5 QuadtreeBuilder.java

```
package quadtree;
import image.ImageUtils;
import compressor.Calculate;
import java.awt.image.BufferedImage;

public class QuadtreeBuilder {
    public static boolean shouldSplit(int[][][] block, double threshold, int minSize,
int method, int[][][] originalBlock) {
        if (block.length <= minSize || block[0].length <= minSize) {
            return false;
        }
```

```
        int[] avgColor = Calculate.calculateAverageColor(block);
        switch (method) {
            case 1:
                return Calculate.calculateVariance(block, avgColor)[3] > threshold;
            case 2:
                return Calculate.calculateMAD(block, avgColor)[3] > threshold;
            case 3:
                return Calculate.calculateMaxPixelDifference(block)[3] > threshold;
            case 4:
                return Calculate.calculateEntropy(block)[3] > threshold;
            case 5:
                return Calculate.calculateSSIM(block, originalBlock)[3] < threshold;
            default:
                return false;
        }
    }

    public static QuadtreeNode buildQuadtree(int[][][] image, int x, int y, int
width, int height, double threshold, int minSize, int method, int[][][]
originalImage) {
        int[][][] block = ImageUtils.extractBlock(image, x, y, width, height);
        int[][][] originalBlock = ImageUtils.extractBlock(originalImage, x, y, width,
height);
        int[] avgColor = Calculate.calculateAverageColor(block);

        if (!shouldSplit(block, threshold, minSize, method, originalBlock)) {
            return new QuadtreeNode(x, y, width, height, avgColor);
        }

        QuadtreeNode node = new QuadtreeNode(x, y, width, height, avgColor);
        int halfWidth = width / 2;
        int halfHeight = height / 2;

        node.children[0] = buildQuadtree(image, x, y, halfWidth, halfHeight,
threshold, minSize, method, originalImage);
        node.children[1] = buildQuadtree(image, x + halfWidth, y, width - halfWidth,
halfHeight, threshold, minSize, method, originalImage);
        node.children[2] = buildQuadtree(image, x, y + halfHeight, halfWidth, height
- halfHeight, threshold, minSize, method, originalImage);
        node.children[3] = buildQuadtree(image, x + halfWidth, y + halfHeight, width
- halfWidth, height - halfHeight, threshold, minSize, method, originalImage);
```

```
        return node;
    }


    public static int calculateTreeDepth(QuadtreeNode node) {
        if (node == null) {
            return 0;
        }

        if (node.children[0] == null) {
            return 1;
        }

        int maxDepth = 0;
        for (QuadtreeNode child : node.children) {
            int depth = calculateTreeDepth(child);
            if (depth > maxDepth) {
                maxDepth = depth;
            }
        }
        return maxDepth + 1;
    }

    public static void reconstructImageByDepth(QuadtreeNode node, int[][][] image,
int originalWidth, int originalHeight, int currentDepth) {
        if (node == null) {
            return;
        }

        if (currentDepth == 1 || node.children[0] == null) {
            for (int i = 0; i < node.height; i++) {
                for (int j = 0; j < node.width; j++) {
                    if (node.y + i < originalHeight && node.x + j < originalWidth) {
                        image[node.y + i][node.x + j] = node.avgColor;
                    }
                }
            }
            return;
        }
```

```
        boolean hasValidChildren = false;
        for (QuadtreeNode child : node.children) {
            if (child != null) {
                hasValidChildren = true;
                reconstructImageByDepth(child, image, originalWidth, originalHeight,
currentDepth - 1);
            }
        }

        if (!hasValidChildren) {
            for (int i = 0; i < node.height; i++) {
                for (int j = 0; j < node.width; j++) {
                    if (node.y + i < originalHeight && node.x + j < originalWidth) {
                        image[node.y + i][node.x + j] = node.avgColor;
                    }
                }
            }
        }
    }


    public static int calculateNodeCount(QuadtreeNode node) {
        if (node == null) {
            return 0;
        }

        int count = 1;
        if (node.children[0] != null) {
            for (QuadtreeNode child : node.children) {
                count += calculateNodeCount(child);
            }
        }
        return count;
    }

    public static BufferedImage convertToBufferedImage(int[][][] pixelData) {
        int height = pixelData.length;
        int width = pixelData[0].length;
        BufferedImage image = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
```

```
        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int r = pixelData[y][x][0];
                int g = pixelData[y][x][1];
                int b = pixelData[y][x][2];
                int rgb = (r << 16) | (g << 8) | b;
                image.setRGB(x, y, rgb);
            }
        }

        return image;
    }
}
```

3.2.5 QuadtreeNode.java

```
package quadtree;

public class QuadtreeNode {
    public int x, y;
    public int width, height;
    public int[] avgColor;
    public QuadtreeNode[] children;

    public QuadtreeNode(int x, int y, int width, int height, int[] avgColor) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.avgColor = avgColor;
        this.children = new QuadtreeNode[4];
    }
}
```

BAB IV

HASIL PROGRAM

4.1. Gambar hitam-putih dengan *threshold* tinggi menggunakan metode 1

input:

```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
9\test\bnw.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
4. Entropy
Masukkan nomor metode: 1
Masukkan ambang batas: 10
Masukkan ukuran blok minimum: 16
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-
2-Stima\Tucil2_13523047_13523139\test\bnw1.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_135
23047_13523139\test\bnw1.GIF
```

output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 222 ms
Ukuran gambar sebelum: 35622 bytes
Ukuran gambar setelah: 21287 bytes
Persentase kompresi: 40.24 %
Kedalaman pohon: 7
Banyak simpul pada pohon: 3001
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\bnw1.GIF
```

Gambar sumber:

Gambar hasil:

4.2. Gambar .jpg penuh warna menjadi .png menggunakan metode 2

input:

```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13
523139\test\bronx.png
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 2
Masukkan ambang batas: 10
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tuci
l-2-Stima\Tucil2_13523047_13523139\test\bronx2.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_1
3523047_13523139\test\bronx2.GIF
```

output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 1488 ms
Ukuran gambar sebelum: 1394357 bytes
Ukuran gambar setelah: 136182 bytes
Persentase kompresi: 90.23 %
Kedalaman pohon: 9
Banyak simpul pada pohon: 18413
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\bronx2.GIF
```

Gambar sumber:

Gambar hasil:

4.3. Gambar dengan detail kecil (bulu hewan) menggunakan metode 3

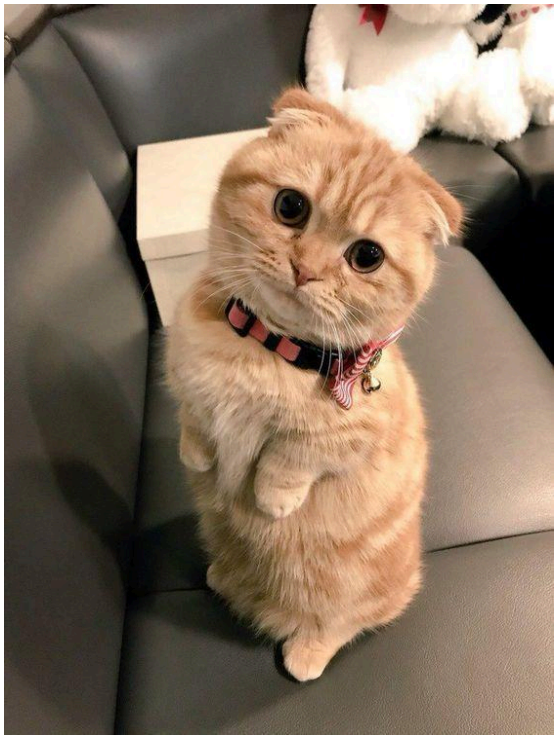
input:

```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139
\test\furry.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 3
Masukkan ambang batas: 39
Masukkan ukuran blok minimum: 7
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-2-St
ima\Tucil2_13523047_13523139\test\furry3.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_1352304
7_13523139\test\furry3.GIF
```

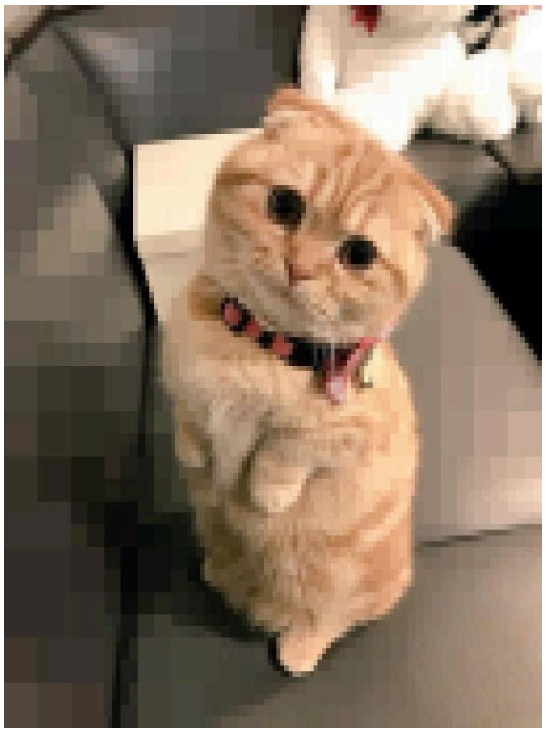
output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 484 ms
Ukuran gambar sebelum: 57216 bytes
Ukuran gambar setelah: 40623 bytes
Persentase kompresi: 29.00 %
Kedalaman pohon: 8
Banyak simpul pada pohon: 9729
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\furry3.GIF
```

Gambar sumber:



Gambar hasil:



4.4. Gambar gradasi dengan threshold tinggi menggunakan metode 4

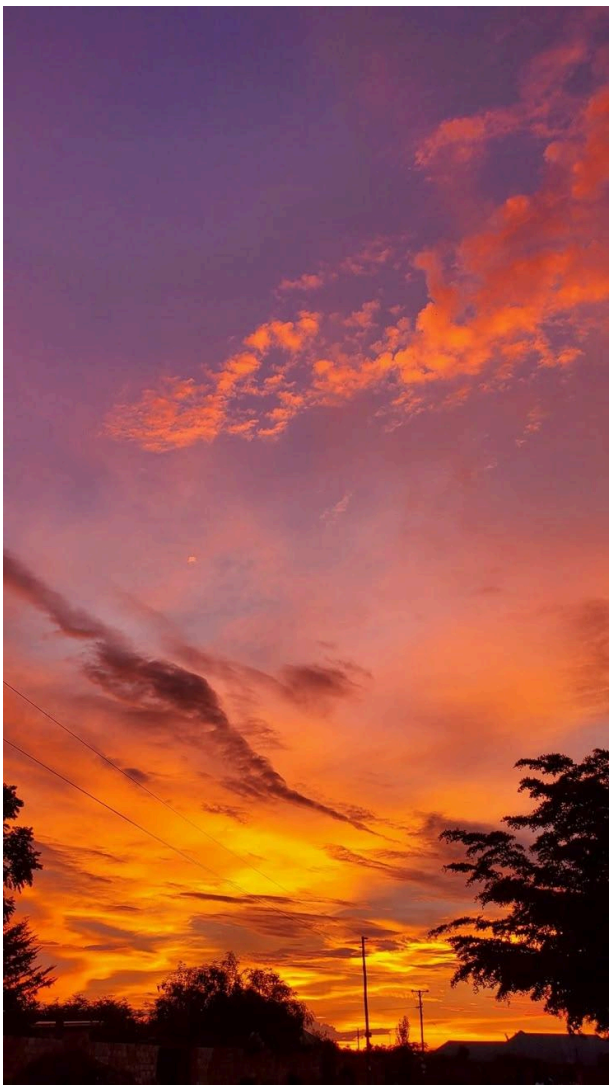
input:

```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139
\test\gradasi.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 4
Masukkan ambang batas: 6
Masukkan ukuran blok minimum: 8
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-2-St
ima\Tucil2_13523047_13523139\test\gradasi4.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_1352304
7_13523139\test\gradasi4.GIF
```

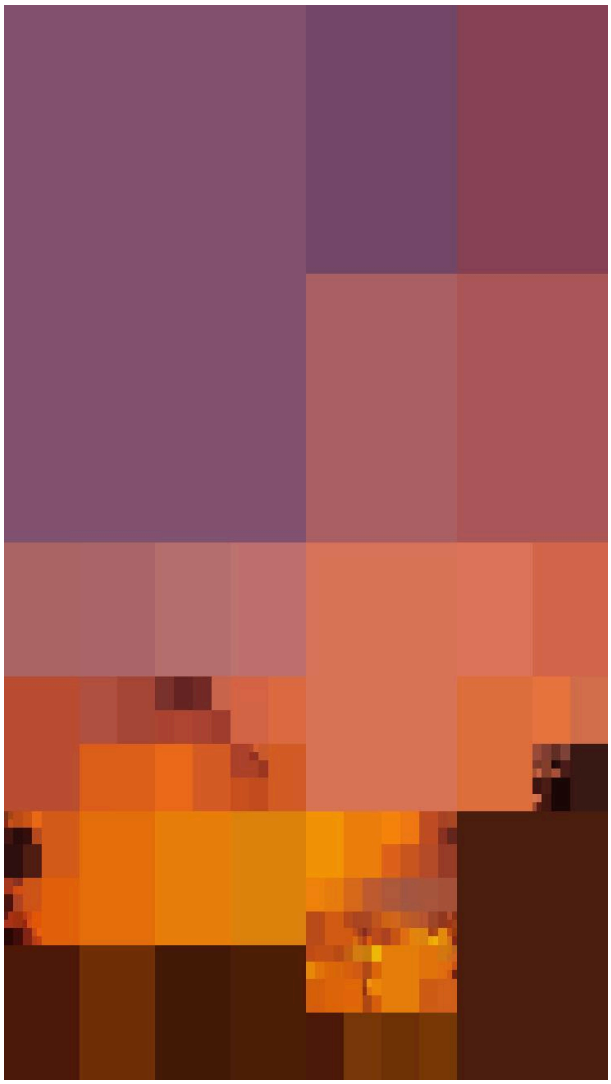
output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 887 ms
Ukuran gambar sebelum: 82499 bytes
Ukuran gambar setelah: 26189 bytes
Persentase kompresi: 68.26 %
Kedalaman pohon: 8
Banyak simpul pada pohon: 293
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\gradasi4.GIF
```

Gambar sumber:



Gambar hasil:



4.5. Gambar menggunakan metode 4

input:

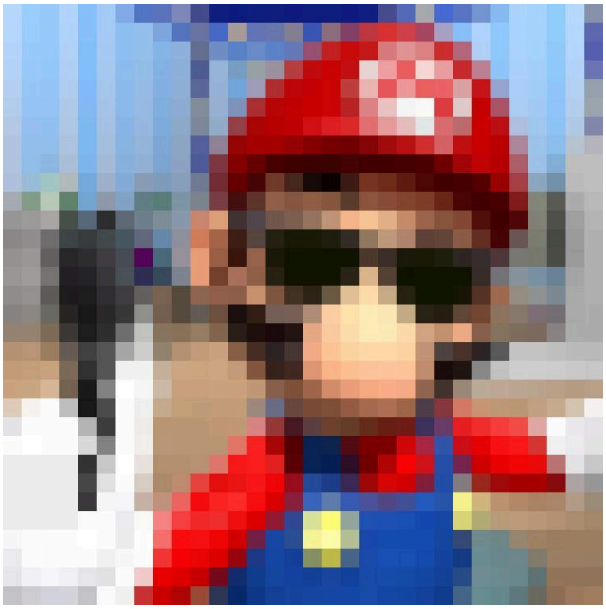
```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139
\test\mario.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 4
Masukkan ambang batas: 2
Masukkan ukuran blok minimum: 32
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-2-St
ima\Tucil2_13523047_13523139\test\mario4.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_1352304
7_13523139\test\mario4.GIF
```

output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 350 ms
Ukuran gambar sebelum: 60907 bytes
Ukuran gambar setelah: 45947 bytes
Persentase kompresi: 24.56 %
Kedalaman pohon: 6
Banyak simpul pada pohon: 1333
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\mario4.GIF
```

Gambar sumber:

Gambar hasil:



4.6. Gambar bayangan menggunakan metode 3

input:

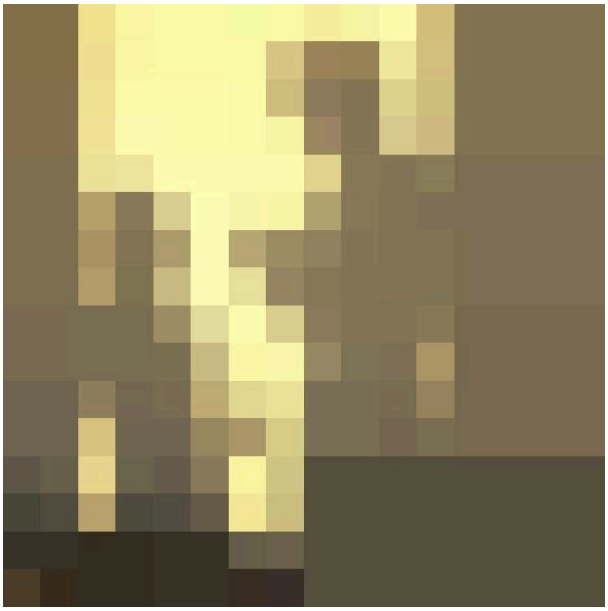
```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139
\test\shadow.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 3
Masukkan ambang batas: 78
Masukkan ukuran blok minimum: 64
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-2-St
ima\Tucil2_13523047_13523139\test\shadow3.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_1352304
7_13523139\test\shadow3.GIF
```

output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 171 ms
Ukuran gambar sebelum: 16939 bytes
Ukuran gambar setelah: 15423 bytes
Persentase kompresi: 8.95 %
Kedalaman pohon: 5
Banyak simpul pada pohon: 189
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\shadow3.GIF
```

Gambar sumber:

Gambar hasil:



4.7. Gambar langit dengan *threshold* tinggi menggunakan metode 2

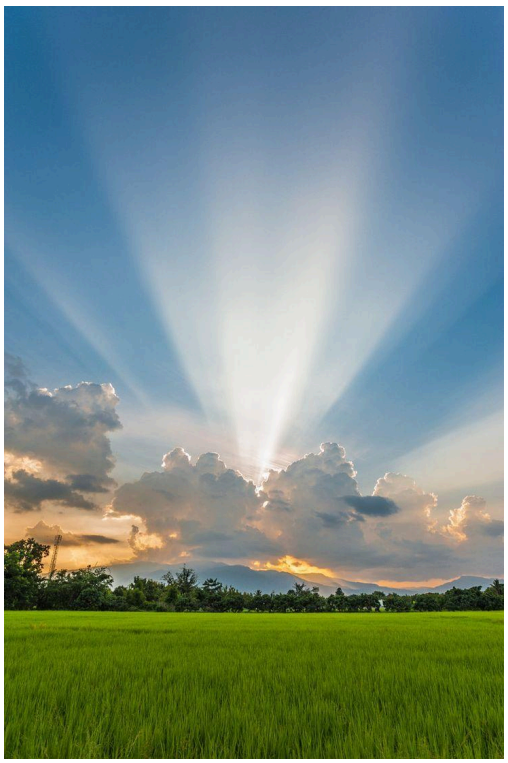
input:

```
PS C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\bin> java Main
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_1352313
9\test\sky.jpg
Pilih metode perhitungan error (1-5):
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan nomor metode: 2
Masukkan ambang batas: 20
Masukkan ukuran blok minimum: 55
Masukkan alamat absolut gambar hasil kompresi (termasuk nama file dan ekstensi): C:\Users\62812\Stima\Tucil-2-S
tima\Tucil2_13523047_13523139\test\sky2.jpg
Masukkan alamat absolut GIF (bonus, kosongkan jika tidak ada): C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_135230
47_13523139\test\sky2.GIF
```

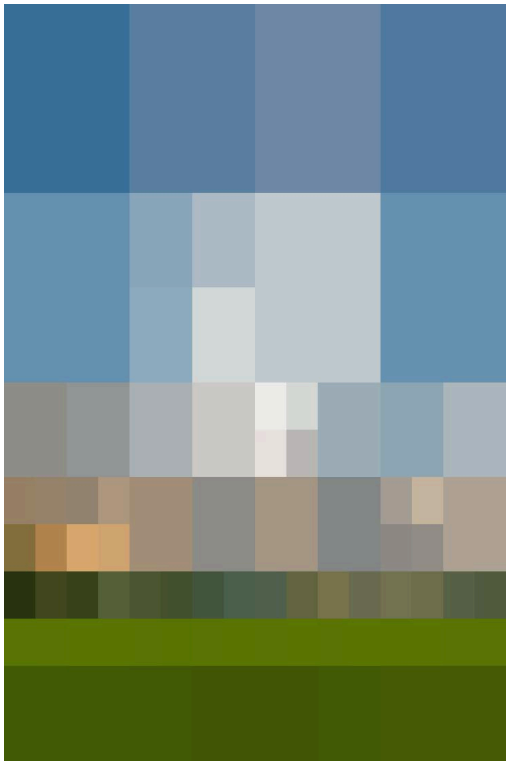
output:

```
Gambar berhasil dibaca menggunakan program Java.
Waktu eksekusi: 740 ms
Ukuran gambar sebelum: 79754 bytes
Ukuran gambar setelah: 22597 bytes
Persentase kompresi: 71.67 %
Kedalaman pohon: 5
Banyak simpul pada pohon: 105
GIF berhasil dibuat: C:\Users\62812\Stima\Tucil-2-Stima\Tucil2_13523047_13523139\test\sky2.GIF
```

Gambar sumber:



Gambar hasil:



BAB V
ANALISIS PERCOBAAN

Algoritma ini menggunakan pendekatan ***divide and conquer*** untuk membagi gambar menjadi blok-blok kecil berdasarkan tingkat keseragaman warna. Jika sebuah blok memenuhi kriteria keseragaman (berdasarkan metode seperti ***Variance***, ***MAD***, ***Max Pixel Difference***, atau ***Entropy***), maka blok tersebut tidak akan dipecah lebih lanjut. Sebaliknya, jika tidak memenuhi kriteria, blok akan dibagi menjadi empat sub-blok hingga mencapai ukuran minimum atau keseragaman terpenuhi.

Kompleksitas Pembangunan Quadtree

- **Proses Pembagian Blok**:
    - Setiap blok dibagi menjadi empat sub-blok. Jika gambar berukuran N x N, maka jumlah blok pada level ke-k adalah 4^k.
    - Kedalaman maksimum pohon adalah $\log_4(N/\text{minSize})$, di mana minSize adalah ukuran blok minimum.
- **Operasi pada Setiap Blok**:
  Untuk setiap blok, algoritma menghitung rata-rata warna dan nilai keseragaman, yang membutuhkan iterasi seluruh piksel dalam blok ($O(N^2)$ untuk blok besar).
- **Total Kompleksitas**:
  Kompleksitas total pembangunan Quadtree adalah **$O(N^2 \log(N))$**.

Kompleksitas Rekonstruksi Gambar

- Rekonstruksi dilakukan dengan traversal pohon Quadtree. Setiap simpul mewakili satu blok pada gambar akhir.
- Kompleksitas traversal adalah **O(M)**, di mana M adalah jumlah simpul dalam Quadtree. Dalam kasus terburuk, $M \approx N^2$, sehingga kompleksitasnya adalah **$O(N^2)$**.

BAB VI
BONUS

Kelas GifSequenceWriter.java merupakan kelas untuk membuat file GIF dari serangkaian gambar (*frame*). Tiap *frame* yang ada pada GIF adalah tahapan kompresi gambar berdasarkan kedalamannya. Apabila sebuah node tidak lagi dibagi, di kedalaman berikutnya, node tersebut akan tetap utuh.

Meskipun kelas ini tidak secara eksplisit menerapkan algoritma kompresi sendiri, proses kompresi tetap terjadi secara otomatis karena format GIF secara bawaan menggunakan kompresi lossless. Selain itu, GIF membatasi warna maksimal menjadi 256 warna (8-bit indexed color), yang secara signifikan mengurangi ukuran gambar dengan cara melakukan kuantisasi warna. Dalam kelas ini, proses dimulai dengan inisialisasi ImageWriter untuk format GIF, diikuti pengaturan metadata seperti delay antar frame dan opsi looping animasi. Metadata ini disimpan secara efisien dalam header GIF.

Saat frame ditambahkan dengan metode **`writeToSequence()`**, setiap gambar dikompresi secara otomatis oleh encoder GIF internal Java. Akhirnya, metode **`close()`** digunakan untuk menyelesaikan proses penulisan GIF.

BAB VII
KESIMPULAN DAN SARAN

7.1 Kesimpulan

Penerapan metode Quadtree dengan strategi Divide and Conquer pada kompresi gambar merupakan pendekatan efektif untuk menyederhanakan representasi citra tanpa kehilangan struktur utamanya. Melalui pembagian gambar menjadi blok-blok kecil secara rekursif, program ini mampu mengevaluasi homogenitas setiap blok berdasarkan metrik statistik seperti variance, mean absolute deviation (MAD), pixel difference, dan entropy. Masing-masing metode memiliki kelebihan dalam mengenali tingkat keseragaman warna, dan pilihan threshold yang tepat sangat menentukan kualitas hasil kompresi.

Pendekatan ini menunjukkan bahwa blok-blok yang dianggap seragam dapat direpresentasikan dengan satu warna, sehingga mengurangi kompleksitas visual gambar dan ukuran penyimpanannya. Di sisi lain, blok yang memiliki banyak detail tetap dipertahankan dalam bentuk subdivisi untuk menjaga akurasi citra. Proses ini secara langsung mencerminkan prinsip Divide and Conquer: membagi gambar menjadi bagian-bagian lebih kecil, menyelesaikan secara lokal, lalu menggabungkannya untuk hasil global yang efisien.

7.2 Saran

Masih terdapat kasus-kasus bonus yang tidak ditangani dan juga terdapat beberapa kasus spesial yang belum dapat diatasi. Oleh karena itu, sebaiknya dikaji lebih lanjut mengenai kasus-kasus spesial ataupun kasus tertentu meskipun tidak dijabarkan pada spesifikasi.

Pada pembuatan program ini melalui Github, masih belum diterapkan best practice sehingga mungkin terjadi miskomunikasi dan konflik kode. Di kesempatan selanjutnya, sebaiknya gunakan Github dengan optimal.

LAMPIRAN

- Tautan Repository GitHub : https://github.com/indahtangdililing/Tucil2_13523047_13523139/

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