IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode *Quadtree*

Laporan Tugas Kecil

Disusun untuk memenuhi tugas besar mata kuliah IF2211 Strategi Algoritma pada Semester II Tahun Akademik 2024/2025

Oleh

Dita Maheswari 13523125

Muhammad Aulia Azka 13523137

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2024

DAFTAR ISI

BAB 1 Analisis Algoritma Divide and Conquer pada Kompresi Gambar Menggunakan QuadTree

Algoritma Divide and Conquer adalah teknik pemecahan masalah dengan membagi masalah menjadi submasalah yang lebih kecil. Submasalah tersebut kemudian diselesaikan secara terpisah, lalu digabungkan untuk mendapatkan solusi akhir. Langkah-langkah algoritma Divide and Conquer adalah:

- Divide: Membagi masalah menjadi submasalah yang lebih kecil dan memiliki kemiripan dengan masalah semula.
- Conquer: Menyelesaikan masing-masing submasalah secara rekursif.
- Combine: Menggabungkan solusi masing-masing submasalah untuk mendapatkan solusi akhir.

Algoritma *Divide and Conquer* bekerja Dengan cara membagi gambar menjadi empat sub-blok secara rekursif hingga setiap blok tidak dapat dibagi - bagi lagi. Langkah pertama dari implementasi algoritma *Divide and Conquer* Quadtree adalah inisiasi awal. Pada awalnya seluruh gambar direpresentasikan sebagai sebuah node akar yang mencakup seluruh area gambar.

Langkah kedua, proses rekursif dilakukan. Pada langkah ini dilakukan pengecekan kelayakan pembagian blok. Setiap node dicek dengan cara memanggil *method* shouldDivide(node). Jika kondisi terpenuhi, method divideNode(node) dipanggil untuk membagi node menjadi empat sub-block. Setelah pembagian, algoritma memanggil kembali fungsi buildQuadTree() secara rekursif untuk keempat node tersebut. Dengan begitu, setiap sub-blok akan dicek apakah perlu dibagi lagi atau tidak.

Langkah ketiga, jika sebuah node tidak memenuhi syarat untuk dibagi, maka node tersebut dianggap sebagai *leaf* node. Pada *leaf* node fungsi processLeafNode(node) dipanggil untuk melakukan proses akhir pada node. Algoritma juga memperbarui statistik seperti maxDepth dan nodeCount.

Langkah keempat, setelah quadtree selesai, gambar yang sudah dikompresi dapat direkonstruksi dari pohon tersebut dengan cara memanggil fungsi reconstructImage() yang berjalan secara rekursif. Jika node adalah *leaf*, maka untuk setiap piksel dalam area yang diwakili node tersebut, set nilai warnanya ke nilai rata - rata yang telah dihitung di node. Jika node bukan leaf, maka fungsi rekursif dipanggil pada keempat anaknya untuk dilakukan proses yang sama.

BAB 2 *Source Code*

2.1 ImageProcessing.java

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

public class ImageProcessing {
    private Color[][] matriksImage;
    private int width;
    private int height;

    // Constructor untuk seluruh gambar
    public ImageProcessing(BufferedImage image) {
        this.width = image.getWidth();
        this.height = image.getHeight();
        matriksImage = new Color[height][width];

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                matriksImage[y][x] = new Color(image.getRGB(x, y));
            }
        }
    }

    // Constructor untuk region gambar
    public ImageProcessing(BufferedImage image, int startX, int startY, int
regionWidth, int regionHeight) {
        this.width = Math.min(regionWidth, image.getWidth() - startX);
        this.height = Math.min(regionHeight, image.getHeight() - startY);
        matriksImage = new Color[height][width];

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                if (startX + x < image.getWidth() && startY + y < image.getHeight())
{
                    matriksImage[y][x] = new Color(image.getRGB(startX + x, startY +
y));
                } else {
                    matriksImage[y][x] = Color.BLACK; // Default jika di luar gambar
                }
            }
        }
    }

    // Getter
```

```
    public Color getPixel(int y, int x) {
        return matriksImage[y][x];
    }

    public int getHeight() {
        return height;
    }

    public int getWidth() {
        return width;
    }

    public Color avgColor(int startX, int startY, int blockWidth, int blockHeight) {
        long sumR = 0;
        long sumG = 0;
        long sumB = 0;

        int ctr = 0;
        for (int y = startY; y < startY + blockHeight && y < height; y++) {
            for (int x = startX; x < startX + blockWidth && x < width; x++) {
                Color c = matriksImage[y][x];
                sumR += c.getRed();
                sumG += c.getGreen();
                sumB += c.getBlue();
                ctr++;
            }
        }

        if (ctr == 0) {
            return new Color(0, 0, 0);
        }

        return new Color((int)(sumR / ctr), (int)(sumG / ctr), (int)(sumB / ctr));
    }
}
```

2.2 ErrorMethods.java

```
package ErrorMethods;

// ini library buat testing
// import java.awt.image.BufferedImage;
// import java.io.File;
// import java.io.IOException;
// import javax.imageio.ImageIO;

import ImageProcessing.ImageProcessing;
```

```java
import java.awt.Color;

public class ErrorMethods {

    private static final double k1 = 0.01;
    private static final double k2 = 0.03;
    private static final double L = 255;

    public enum ErrorMethodType {
        MAD, MAX_PIXEL_DIFF, ENTROPY, VARIANCE, SSIM
    }

    // ubah blok gambar jadi greyscale
    private static int[][] toGrayscaleMatrix(ImageProcessing imgProc) {
        int height = imgProc.getHeight();
        int width = imgProc.getWidth();
        int[][] grayscale = new int[height][width];

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                Color pixel = imgProc.getPixel(y, x);
                grayscale[y][x] = (int) (0.299 * pixel.getRed() + 0.587 *
pixel.getGreen() + 0.114 * pixel.getBlue());
            }
        }
        return grayscale;
    }

    // MEAN ABSOLUTE DEVIATION (MAD)
    public static double mad(ImageProcessing imgProc) {
        int[][] block = toGrayscaleMatrix(imgProc);
        int N = block.length * block[0].length;
        double mean = 0;

        for (int[] row : block) {
            for (int pixel : row) {
                mean += pixel;
            }
        }
        mean /= N;

        double mad = 0;
        for (int[] row : block) {
            for (int pixel : row) {
                mad += Math.abs(pixel - mean);
            }
        }
        return mad / N;
```

```java
    }

    // MAX PIXEL DIFFERENCE
    public static double maxPixelDiff(ImageProcessing imgProc) {
        int[][] block = toGrayscaleMatrix(imgProc);
        int maxVal = Integer.MIN_VALUE;
        int minVal = Integer.MAX_VALUE;

        for (int[] row : block) {
            for (int pixel : row) {
                if (pixel > maxVal) {
                    maxVal = pixel;
                }
                if (pixel < minVal) {
                    minVal = pixel;
                }
            }
        }
        return maxVal - minVal;
    }

    // ENTROPY
    public static double entropy(ImageProcessing imgProc) {
        int[][] block = toGrayscaleMatrix(imgProc);
        int[] histogram = new int[256];
        int totalPixels = block.length * block[0].length;

        for (int[] row : block) {
            for (int pixel : row) {
                histogram[pixel]++;
            }
        }

        double ent = 0;
        for (int freq : histogram) {
            if (freq > 0) {
                double prob = (double) freq / totalPixels;
                ent -= prob * (Math.log(prob) / Math.log(2));
            }
        }
        return ent;
    }
    // VARIANCE
    public static double variance(ImageProcessing imgProc) {
        int height = imgProc.getHeight();
        int width = imgProc.getWidth();
        int N = height * width;
```

```
        double meanR = 0, meanG = 0, meanB = 0;
        double varR = 0, varG = 0, varB = 0;

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                Color pixel = imgProc.getPixel(y, x);
                meanR += pixel.getRed();
                meanG += pixel.getGreen();
                meanB += pixel.getBlue();
            }
        }

        meanR /= N;
        meanG /= N;
        meanB /= N;

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                Color pixel = imgProc.getPixel(y, x);
                varR += Math.pow(pixel.getRed() - meanR, 2);
                varG += Math.pow(pixel.getGreen() - meanG, 2);
                varB += Math.pow(pixel.getBlue() - meanB, 2);
            }
        }

        varR /= N;
        varG /= N;
        varB /= N;

        return (varR + varG + varB) / 3;
    }



    // SSIM
    public static double ssimRGB(ImageProcessing ori, ImageProcessing comp){
        if (ori.getWidth() != comp.getWidth() || ori.getHeight() !=
comp.getHeight()){
            throw new IllegalArgumentException("Ukuran blok tidak sama untuk
perhitungan SSIM.");
        }

        double ssimR = ssimChannel(ori, comp, 'R');
        double ssimG = ssimChannel(ori, comp, 'G');
        double ssimB = ssimChannel(ori, comp, 'B');

        double wR = 1.0 / 3.0;
        double wG = 1.0 / 3.0;
```

```
        double wB = 1.0 / 3.0;

        return wR * ssimR + wG * ssimG + wB * ssimB;
    }

    private static double ssimChannel(ImageProcessing ori, ImageProcessing comp,
char channel){
        int height = ori.getHeight();
        int width = ori.getWidth();

        int N = width * height;

        double sumO = 0.0, sumC = 0.0;
        double sumO2 = 0.0, sumC2 = 0.0;
        double sumOC = 0.0;

        for (int y = 0; y < height; y++){
            for (int x = 0; x < width; x++){
                Color co = ori.getPixel(y, x);
                Color cc = comp.getPixel(y, x);

                int vo = (channel == 'R') ? co.getRed() :
                (channel == 'G') ? co.getGreen() : co.getBlue();

                int vc = (channel == 'R') ? cc.getRed() :
                (channel == 'G') ? cc.getGreen() : cc.getBlue();

                sumO += vo;
                sumC += vc;
                sumO2 += (vo * (double) vo);
                sumC2 += (vc * (double) vc);
                sumOC += (vo * (double) vc);
            }
        }

        double meanO = sumO / N;
        double meanC = sumC / N;

        double varO = (sumO2 / N) - (meanO * meanO);
        double varC = (sumC2 / N) - (meanC * meanC);

        double covOC = (sumOC / N) - (meanO * meanC);

        double c1 = (k1 * L) * (k1 * L);
        double c2 = (k2 * L) * (k2 * L);

        double numerator   = (2 * meanO * meanC + c1) * (2 * covOC + c2);
        double denominator = (meanO * meanO + meanC * meanC + c1) * (varO + varC +
```

```
c2);

        return numerator / denominator;
    }
}
```

2.3 BlockDivision.java

```
// Pembagian Blok
// Bandingkan nilai variansi blok dengan threshold

package BlockDivision;

import ColorNormalizer.ColorNormalizer;
import ErrorMethods.ErrorMethods;
import ErrorMethods.ErrorMethods.ErrorMethodType;
import ImageProcessing.ImageProcessing;
import QuadTree.QuadTreeNode;
import java.awt.Color;
import java.awt.image.BufferedImage;

public class BlockDivision {
    private int minBlockSize;
    private double threshold;
    private BufferedImage image;
    private ColorNormalizer colorNormalizer;
    private ErrorMethodType errorMethodType;

    // konstruktor untuk inisialisasi block division
    public BlockDivision(int minBlockSize, double threshold, ErrorMethodType
errorMethodType, BufferedImage image) {
        this.minBlockSize = minBlockSize;
        this.threshold = threshold;
        this.errorMethodType = errorMethodType;
        this.image = image;
        this.colorNormalizer = new ColorNormalizer(image);
    }

    // menentukan apakah sebuah node perlu dibagi berdasarkan error methods
    public boolean shouldDivided(QuadTreeNode node) {
        ImageProcessing imgProc = new ImageProcessing(image, node.getX(),
node.getY(), node.getWidth(), node.getHeight());

        // jika ukuran blok sudah minimum, jangan dibagi lagi
        if (imgProc.getWidth() <= minBlockSize || imgProc.getHeight() <=
minBlockSize) {
            return false;
```

```
        }

        double err = 0;

        switch (errorMethodType) {
            case MAD:
                err = ErrorMethods.mad(imgProc);
                break;
            case MAX_PIXEL_DIFF:
                err = ErrorMethods.maxPixelDiff(imgProc);
                break;
            case ENTROPY:
                err = ErrorMethods.entropy(imgProc);
                break;
            case VARIANCE:
                err = ErrorMethods.variance(imgProc);
                break;
            case SSIM:
                {
                    Color avgColor = imgProc.avgColor(0, 0, imgProc.getWidth(),
imgProc.getHeight());

                    BufferedImage compressedBlockImage = new
BufferedImage(imgProc.getWidth(), imgProc.getHeight(),
BufferedImage.TYPE_INT_RGB);
                    for (int y = 0; y < imgProc.getHeight(); y++){
                        for (int x = 0; x < imgProc.getWidth(); x++){
                            compressedBlockImage.setRGB(x, y, avgColor.getRGB());
                        }
                    }

                    ImageProcessing compressedBlock = new
ImageProcessing(compressedBlockImage);

                    err = ErrorMethods.ssimRGB(imgProc, compressedBlock);

                    return err < threshold;
                }
        }

        return err > threshold;
    }

    public void divideNode(QuadTreeNode node) {
        int halfWidth = node.getWidth() / 2;
        int halfHeight = node.getHeight() / 2;

        // buat 4 anak node dengan ukuran setengah dari node saat ini
```

```
        QuadTreeNode topLeft = new QuadTreeNode(node.getX(), node.getY(),
halfWidth, halfHeight, node.getDepth() + 1);
        QuadTreeNode topRight = new QuadTreeNode(node.getX() + halfWidth,
node.getY(), node.getWidth() - halfWidth, halfHeight, node.getDepth() + 1);
        QuadTreeNode bottomLeft = new QuadTreeNode(node.getX(), node.getY() +
halfHeight, halfWidth, node.getHeight() - halfHeight, node.getDepth() + 1);
        QuadTreeNode bottomRight = new QuadTreeNode(node.getX() + halfWidth,
node.getY() + halfHeight, node.getWidth() - halfWidth, node.getHeight() -
halfHeight, node.getDepth() + 1);

        //gunakan setter untuk menetapkan anak node
        node.setChildren(topLeft, topRight, bottomLeft, bottomRight);
    }

    public void processLeafNode(QuadTreeNode node) {
        colorNormalizer.normalizeColor(node);
    }

    public void setThreshold(double threshold) {
        this.threshold = threshold;
    }

    public double getThreshold() {
        return threshold;
    }
}
```

2.4 ColorNormalizer.java

```
// Normalisasi Warna
// Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai
dengan
// rata-rata nilai RGB blok.


package ColorNormalizer;

import QuadTree.QuadTreeNode;
import java.awt.Color;
import java.awt.image.BufferedImage;
// import QuadTree.QuadTreeBuilder;

public class ColorNormalizer {
    private BufferedImage image;

    public ColorNormalizer(BufferedImage image) {
        this.image = image;
```

```
    }

    // normalisasi warna untuk setiap node dalam QuadTree
    public void normalizeColor(QuadTreeNode node) {
        int totR = 0;
        int totG = 0;
        int totB = 0;
        int pixelCount = 0;

        for(int y = node.getY(); y < node.getY() + node.getHeight() && y <
image.getHeight(); y++) {
            for (int x = node.getX(); x < node.getX() + node.getWidth() && x <
image.getWidth(); x++) {
                Color color = new Color(image.getRGB(x, y));
                totR += color.getRed();
                totG += color.getGreen();
                totB += color.getBlue();
                pixelCount++;
            }
        }

        if(pixelCount > 0) {
            int avgR = totR / pixelCount;
            int avgG = totG / pixelCount;
            int avgB = totB / pixelCount;
            node.setColor(new Color(avgR, avgG, avgB));
        } else {
            node.setColor(Color.BLACK);
        }
    }
}
```

2.5 QuadTreeBuilder.java

```
// Rekursi dan Penghentian
// Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga
semua
// blok memenuhi salah satu dari dua kondisi

package QuadTree;

import BlockDivision.BlockDivision;
import java.awt.image.BufferedImage;

public class QuadTreeBuilder {
    private BufferedImage oriImage;
    private BlockDivision blockDivision;
```

```
    private QuadTreeNode root;
    private int maxDepth = 0;
    private int nodeCount = 0;

    public QuadTreeBuilder(BufferedImage oriImage, BlockDivision blockDivision) {
        this.oriImage = oriImage;
        this.blockDivision = blockDivision;
    }

    // buat quadtree dari gambar
    public void buildTree() {
        root = new QuadTreeNode(0, 0, oriImage.getWidth(), oriImage.getHeight(),
0);
        buildQuadTree(root);
    }

    // fungsi rekursif untuk membuat quadtree
    private void buildQuadTree(QuadTreeNode node) {
        if(blockDivision.shouldDivided(node)) {
            blockDivision.divideNode(node);

            // buat 4 anak node secara rekursif
            buildQuadTree(node.getTopLeft());
            buildQuadTree(node.getTopRight());
            buildQuadTree(node.getBottomLeft());
            buildQuadTree(node.getBottomRight());
        }
        else {
            blockDivision.processLeafNode(node);
            maxDepth = Math.max(maxDepth, node.getDepth());
            nodeCount++;
        }
    }

    // Buat kembali gambar dari quadtree - NOTE: Ada bug typo di perulangan for
    private void reconstructImage(QuadTreeNode node, BufferedImage image) {
        if (node == null) return;

        if (node.isLeaf()) {
            for (int y = node.getY(); y < node.getY() + node.getHeight() && y <
image.getHeight(); y++) {
                for (int x = node.getX(); x < node.getX() + node.getWidth() && x <
image.getWidth(); x++) { // Bug diperbaiki di sini, sebelumnya "y++"
                    image.setRGB(x, y, node.getColor().getRGB());
                }
            }
        }
        else {
```

```
            // buat gambar untuk 4 anak node secara rekursif
            reconstructImage(node.getTopLeft(), image);
            reconstructImage(node.getTopRight(), image);
            reconstructImage(node.getBottomLeft(), image);
            reconstructImage(node.getBottomRight(), image);
        }
    }
    //getters
    public QuadTreeNode getRoot() {
        return root;
    }

    public int getMaxDepth() {
        return maxDepth;
    }

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

2.6 QuadTreeNode.java

```
package QuadTree;

import ImageProcessing.ImageProcessing;
import java.awt.Color;

public class QuadTreeNode {
    int x, y, width, height, depth;
    Color color;
    QuadTreeNode topLeft, topRight, bottomLeft, bottomRight;
    private ImageProcessing imgProc;

    public QuadTreeNode(int x, int y, int width, int height, int depth) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.depth = depth;
        this.color = Color.BLACK;
    }

    public void setChildren(QuadTreeNode topLeft, QuadTreeNode topRight,
QuadTreeNode bottomLeft, QuadTreeNode bottomRight) {
        this.topLeft = topLeft;
        this.topRight = topRight;
```

```
        this.bottomLeft = bottomLeft;
        this.bottomRight = bottomRight;
    }

    public boolean isLeaf() {
        return topLeft == null && topRight == null && bottomLeft == null &&
bottomRight == null;
    }

    // getters
    public int getX() { return x; }
    public int getY() { return y; }
    public int getWidth() { return width; }
    public int getHeight() { return height; }
    public int getDepth() { return depth; }
    public Color getColor() { return color; }
    public QuadTreeNode getTopLeft() { return topLeft; }
    public QuadTreeNode getTopRight() { return topRight; }
    public QuadTreeNode getBottomLeft() { return bottomLeft; }
    public QuadTreeNode getBottomRight() { return bottomRight; }

    public void setColor(Color color) { this.color = color; }

    public ImageProcessing getImageProcessing() {
        return imgProc;
    }

}
```

2.7 OutputImage.java

```
package OutputImage;

import QuadTree.QuadTreeBuilder;
import QuadTree.QuadTreeNode;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;


public class OutputImage {
    private BufferedImage originalImage;
    private QuadTreeNode root;
    private String outputPath;
    private BufferedImage compressedImage;
```

```
    public OutputImage(BufferedImage originalImage, QuadTreeBuilder quadTree,
String outputPath) {
        this.originalImage = originalImage;
        this.root = quadTree.getRoot();
        this.outputPath = outputPath;
        this.compressedImage = new BufferedImage(originalImage.getWidth(),
        originalImage.getHeight(), BufferedImage.TYPE_INT_RGB);
    }


    public void saveCompressedImage() throws IOException {
        try {
            // reconstruct image dari quadtree
            reconstructImage(root, compressedImage);

            // pastikan direktori ada
            File outputFile = new File(outputPath);
            File parentDir = outputFile.getParentFile();
            if (parentDir != null && !parentDir.exists()) {
                if (!parentDir.mkdirs()){
                    System.out.println("Peringatan: Gagal membuat direktori " +
parentDir);
                }
                // boolean dirCreated = parentDir.mkdirs();
                // if (!dirCreated) {
                //     System.out.println("Peringatan: Gagal membuat direktori.");
                // }
            }

            String extension = getExtension(outputPath);

            if (extension == null){
                extension = "png"; // extension default
            }

            extension = extension.toLowerCase();

            boolean success = false;

            if (extension.equals("jpg") || extension.equals("jpeg")){
                success = ImageIO.write(compressedImage, "jpg", outputFile);
            }else if (extension.equals("png")){
                success = ImageIO.write(compressedImage, "png", outputFile);
            }else{
                extension = "png";
                success = ImageIO.write(compressedImage, "png", outputFile);
            }
            // Coba pendekatan alternatif - gunakan File langsung
```

```
            // boolean success = ImageIO.write(compressedImage, "jpg",
outputFile);

            if (!success) {
                // Jika pendekatan pertama gagal, coba pendekatan kedua dengan
format berbeda
                System.out.println("Gagal menyimpan dalam format " + extension +
", mencoba fallback ke PNG...");
                extension = "png";
                success = ImageIO.write(compressedImage, "png", outputFile);

                if (!success) {
                    // Jika pendekatan kedua gagal, coba pendekatan ketiga dengan
FileOutputStream
                    // System.out.println("Format JPG gagal, mencoba dengan
pendekatan FileOutputStream...");
                    // BufferedImage tempImage = new
BufferedImage(compressedImage.getWidth(), compressedImage.getHeight(),
BufferedImage.TYPE_INT_RGB);
                    // Graphics2D g2d = tempImage.createGraphics();
                    // g2d.drawImage(compressedImage, 0, 0, null);
                    // g2d.dispose();

                    // // Coba simpan ke lokasi yang pasti bisa diakses
                    // File tempFile = new File(System.getProperty("user.home") +
"/compressed_image.jpg");
                    // ImageIO.write(tempImage, "jpg", tempFile);
                    // System.out.println("Gambar berhasil disimpan sementara di:
" + tempFile.getAbsolutePath());

                    throw new IOException("Gagal menyimpan gambar dalam format PNG
fallback.");
                    // Coba pindahkan ke lokasi yang diinginkan
                    // try (FileOutputStream fos = new
FileOutputStream(outputFile)) {
                    //     byte[] imageData =
java.nio.file.Files.readAllBytes(tempFile.toPath());
                    //     fos.write(imageData);
                    //     tempFile.delete();
                    // } catch (Exception e) {
                    //     throw new IOException("Gagal menyimpan gambar ke lokasi
akhir: " + e.getMessage());
                    // }
                }
            }

        } catch (Exception e) {
            // Tangkap semua exception untuk debugging
```

```
            System.err.println("Detail error: " + e.getClass().getName() + ": " +
e.getMessage());
            e.printStackTrace();
            throw new IOException("Error saat menyimpan gambar: " +
e.getMessage());
        }
    }


    private void reconstructImage(QuadTreeNode node, BufferedImage image) {
        if (node == null) return;

        if (node.isLeaf()) {

            for (int y = node.getY(); y < node.getY() + node.getHeight() && y <
image.getHeight(); y++) {
                for (int x = node.getX(); x < node.getX() + node.getWidth() && x <
image.getWidth(); x++) {
                    image.setRGB(x, y, node.getColor().getRGB());
                }
            }
        } else {
            // Recursively reconstruct the image from child nodes
            reconstructImage(node.getTopLeft(), image);
            reconstructImage(node.getTopRight(), image);
            reconstructImage(node.getBottomRight(), image);
            reconstructImage(node.getBottomLeft(), image);
        }
    }

    private  String getExtension(String path){
        int dotIndex = path.lastIndexOf('.');

        if (dotIndex == -1 || dotIndex == path.length() - 1){
            return null;
        }

        return path.substring(dotIndex + 1);
    }
}
```

2.8 Main.java

```
import BlockDivision.BlockDivision;
import ErrorMethods.ErrorMethods.ErrorMethodType;
import OutputImage.OutputImage;
import QuadTree.QuadTreeBuilder;
```

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

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

        try {
            // 1. Input alamat gambar asli
            System.out.print("Masukkan alamat absolut gambar yang akan dikompresi:
");
            String inputPath = scanner.nextLine();
            File inputFile = new File(inputPath);
            if (!inputFile.exists()) {
                System.err.println("File tidak ditemukan: " + inputPath);
                return;
            }
            BufferedImage originalImage = ImageIO.read(inputFile);

            // 2. Pilih metode perhitungan error
            int methodChoice = 0;
            boolean validMethodChoice = false;
            while (!validMethodChoice) {
                try {
                    System.out.println("Pilih metode perhitungan error:");
                    System.out.println("1. Mean Absolute Deviation (MAD)");
                    System.out.println("2. Max Pixel Difference");
                    System.out.println("3. Entropy");
                    System.out.println("4. Variance");
                    System.err.println("5. Structural Similarity Index (SSIM)");
                    System.err.println("Masukkan Pilihan (1 - 5): ");
                    methodChoice = Integer.parseInt(scanner.nextLine().trim());
                    if (methodChoice >= 1 && methodChoice <= 5) {
                        validMethodChoice = true;
                    } else {
                        System.out.println("Pilihan tidak valid. Silakan masukkan
angka 1 - 5.");
                    }
                } catch (NumberFormatException e) {
                    System.out.println("Input tidak valid. Silakan masukkan angka
1 - 5.");
                }
            }

            // 3. Input ambang batas
```

```
            double threshold = 0;
            boolean validThreshold = false;
            while (!validThreshold) {
                try {
                    System.out.print("Masukkan ambang batas (sesuai dengan metode
yang dipilih): ");
                    threshold = Double.parseDouble(scanner.nextLine().trim());
                    validThreshold = true;
                } catch (NumberFormatException e) {
                    System.out.println("Input tidak valid. Silakan masukkan nilai
numerik.");
                }
            }

            // 4. Input ukuran blok minimum
            int minBlockSize = 0;
            boolean validBlockSize = false;
            while (!validBlockSize) {
                try {
                    System.out.print("Masukkan ukuran blok minimum: ");
                    minBlockSize = Integer.parseInt(scanner.nextLine().trim());
                    if (minBlockSize > 0) {
                        validBlockSize = true;
                    } else {
                        System.out.println("Ukuran blok harus lebih besar dari
0.");
                    }
                } catch (NumberFormatException e) {
                    System.out.println("Input tidak valid. Silakan masukkan nilai
numerik.");
                }
            }

            // // 5. Input target persentase kompresi
            // double targetCompression = 0;
            // boolean validCompression = false;
            // while (!validCompression) {
            //     try {
            //         System.out.print("Masukkan target persentase kompresi (0
untuk nonaktif): ");
            //         targetCompression =
Double.parseDouble(scanner.nextLine().trim());
            //         if (targetCompression >= 0) {
            //             validCompression = true;
            //         } else {
            //             System.out.println("Persentase kompresi tidak boleh
negatif.");
            //         }
```

```
            //     } catch (NumberFormatException e) {
            //         System.out.println("Input tidak valid. Silakan masukkan
nilai numerik.");
            //     }
            // }

            // 5. Input alamat gambar hasil kompresi
            System.out.print("Masukkan alamat absolut gambar hasil kompresi: ");
            String outputPath = scanner.nextLine().trim();

            // Waktu mulai eksekusi
            long startTime = System.nanoTime();

            // Proses konversi ke ImageProcessing
            // ImageProcessing imgProc = new ImageProcessing(originalImage);

            // Pilih metode error
            ErrorMethodType errorMethodType;
            // double computedThreshold = 0;

            switch (methodChoice) {
                case 1:
                    errorMethodType = ErrorMethodType.MAD;
                    // computedThreshold = ErrorMethods.mad(imgProc);
                    break;
                case 2:
                    errorMethodType = ErrorMethodType.MAX_PIXEL_DIFF;
                    // computedThreshold = ErrorMethods.maxPixelDiff(imgProc);
                    break;
                case 3:
                    errorMethodType = ErrorMethodType.ENTROPY;
                    // computedThreshold = ErrorMethods.entropy(imgProc);
                    break;
                case 4:
                    errorMethodType = ErrorMethodType.VARIANCE;
                    // computedThreshold = ErrorMethods.variance(imgProc);
                    break;
                case 5:
                    errorMethodType = ErrorMethodType.SSIM;
                    break;
                default:
                    throw new IllegalArgumentException("Metode tidak valid!");
            }

            // Jika mode kompresi aktif, sesuaikan threshold
            // if (targetCompression > 0) {
            //     threshold = computedThreshold * targetCompression;
            //     System.out.println("Ambang batas yang disesuaikan: " +
```

```
threshold);
            // }

            // Bangun quadtree
            System.out.println("Membangun quadtree...");
            BlockDivision blockDivision = new BlockDivision(minBlockSize,
threshold, errorMethodType, originalImage);
            QuadTreeBuilder quadTree = new QuadTreeBuilder(originalImage,
blockDivision);
            quadTree.buildTree();

            // Simpan hasil kompresi
            System.out.println("Menyimpan hasil kompresi...");
            OutputImage outputImage = new OutputImage(originalImage, quadTree,
outputPath);
            outputImage.saveCompressedImage();

            // Waktu selesai eksekusi
            long endTime = System.nanoTime();

            // 8. Waktu eksekusi
            double executionTime = (endTime - startTime) / 1e9;
            System.out.println("Waktu eksekusi: " + executionTime + " detik");

            // 9. Ukuran gambar sebelum
            // File originalFile = new File(inputPath);
            long originalSize = inputFile.length();
            System.out.println("Ukuran gambar sebelum: " + originalSize + "
bytes");

            // 10. Ukuran gambar setelah
            File compressedFile = new File(outputPath);
            long compressedSize = compressedFile.length();
            System.out.println("Ukuran gambar setelah: " + compressedSize + "
bytes");

            // 11. Persentase kompresi
            double compressionPercentage = (1 - (double) compressedSize /
originalSize) * 100;
            System.out.println("Persentase kompresi: " + compressionPercentage +
"%");

            // 12. Kedalaman pohon
            System.out.println("Kedalaman pohon: " + quadTree.getMaxDepth());

            // 13. Banyak simpul pada pohon
            System.out.println("Banyak simpul pada pohon: " +
quadTree.getNodeCount());
```

```
            // 14. Gambar hasil kompresi disimpan
            System.out.println("Gambar hasil kompresi telah disimpan di: " +
outputPath);

        } catch (IOException e) {
            System.err.println("Terjadi kesalahan saat membaca atau menyimpan
gambar: " + e.getMessage());
            e.printStackTrace();
        } catch (Exception e) {
            System.err.println("Terjadi kesalahan: " + e.getMessage());
            e.printStackTrace();
        } finally {
            scanner.close();
        }
    }
}
```

BAB 3 Pengujian

Testing yang dilakukan pada bab pengujian ini adalah tes 3 gambar dengan ekstensi yang berbeda yaitu, jpg, png, dan jpeg. Lalu untuk inputnya sendiri antara lain alamat absolut gambar yang akan dikompresi, metode perhitungan error (gunakan penomoran sebagai input), ambang batas, ukuran blok minimum, dan alamat absolut gambar hasil kompresi. Untuk keterangan range ambang batas sebagai berikut:

1. MAD (Mean Absolute Deviation)

Ambang batas yang disarankan: 5 – 20. Nilai MAD berkisar dari 0 (sangat homogen) hingga mendekati 255 (kontras tinggi). Nilai 5 – 20 cocok untuk deteksi kontras sedang. Semakin kecil threshold, semakin sensitif deteksinya dan blok akan lebih sering dibagi.

2. Max Pixel Difference

Ambang batas yang disarankan: 10 – 50, karena nilai ini langsung merepresentasikan selisih pixel tertinggi dan terendah, 0 berarti blok homogen. Nilai tinggi hanya akan memisahkan area dengan perbedaan sangat tajam, menghasilkan kompresi tinggi tapi bisa mengaburkan detail.

3. Entropy

Ambang batas yang disarankan: 2 – 5. Entropy maksimum untuk gambar 8-bit adalah ~8, tapi untuk blok kecil jarang melebihi 6. Semakin kecil threshold, semakin besar kemungkinan blok dianggap "berinformasi tinggi" dan dibagi lebih lanjut.

4. Variance

Ambang batas yang disarankan: 500 – 2000. Variansi memiliki rentang yang luas (karena sifat kuadrat), bergantung pada ukuran dan kontras blok.
Untuk gambar dengan noise, threshold bisa dinaikkan agar tidak memicu pembagian yang berlebihan.

5. SSIM (Structural Similarity Index)

Ambang batas yang disarankan: 0.75 – 0.95. SSIM bernilai antara 0 (tidak mirip) hingga 1 (identik). Threshold SSIM < 0.9 berarti kita hanya menyimpan blok jika perbandingan strukturnya benar-benar bagus. Semakin tinggi threshold, semakin ketat kriteria visual — blok akan dibagi lebih sering untuk mempertahankan kualitas.

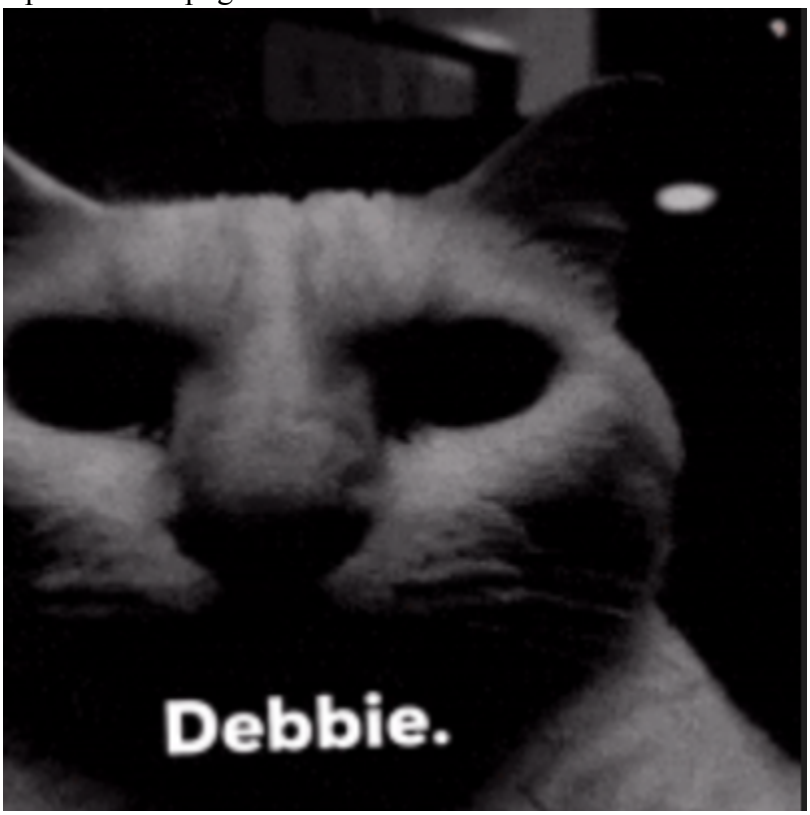
Adapun catatan tambahan untuk ambang batas :

- Nilai-nilai ini dapat berubah tergantung ukuran blok dan resolusi gambar.
- Untuk gambar grayscale, MAD dan Variance biasanya menghasilkan nilai lebih rendah, threshold bisa disesuaikan ke bawah.
- Untuk pengujian awal, disarankan menggunakan gambar dengan area kontras dan datar agar mudah terlihat dampak perubahan threshold.

Untuk bentuk keluaran atau hasil dari pengujiannya sendiri adalah, waktu eksekusi, ukuran gambar sebelum, ukuran gambar setelah, Persentase kompresi, kedalaman pohon, banyak simpul pohon, dan gambar hasil kompresi pada alamat yang sudah ditentukan.

3.1 Pengujian 1

Input Gambar png:



Input Perhitungan Error SSIM:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Debbie.png
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
5
Masukkan ambang batas (sesuai dengan metode yang dipilih): 0.9
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DeggieSolSSIM.png
```

Output:

```
Waktu eksekusi: 0.1671487 detik
Ukuran gambar sebelum: 142221 bytes
Ukuran gambar setelah: 30208 bytes
Persentase kompresi: 78.75981746718136%
Kedalaman pohon: 7
Banyak simpul pada pohon: 9001
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DeggieSolSSIM.png
```

Output Gambar

3.2 Pengujian 2

Input Gambar png:

Input Perhitungan error variance:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Debbie.png
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
4
Masukkan ambang batas (sesuai dengan metode yang dipilih): 1000
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolVar.png
```

Output:

```
Waktu eksekusi: 0.0842443 detik
Ukuran gambar sebelum: 142221 bytes
Ukuran gambar setelah: 9764 bytes
Persentase kompresi: 93.13462850071367%
Kedalaman pohon: 7
Banyak simpul pada pohon: 571
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolVar.png
```

Output Gambar:

3.3 Pengujian 3

Input Gambar png:

Input Perhitungan Error MAD:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Debbie.png
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
1
Masukkan ambang batas (sesuai dengan metode yang dipilih): 10
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolMAD.png
```

Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.1135353 detik
Ukuran gambar sebelum: 142221 bytes
Ukuran gambar setelah: 17169 bytes
Persentase kompresi: 87.92794313076128%
Kedalaman pohon: 7
Banyak simpul pada pohon: 2230
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolMAD.png
```

Output Gambar:

3.4 Pengujian 4

Input Gambar png:

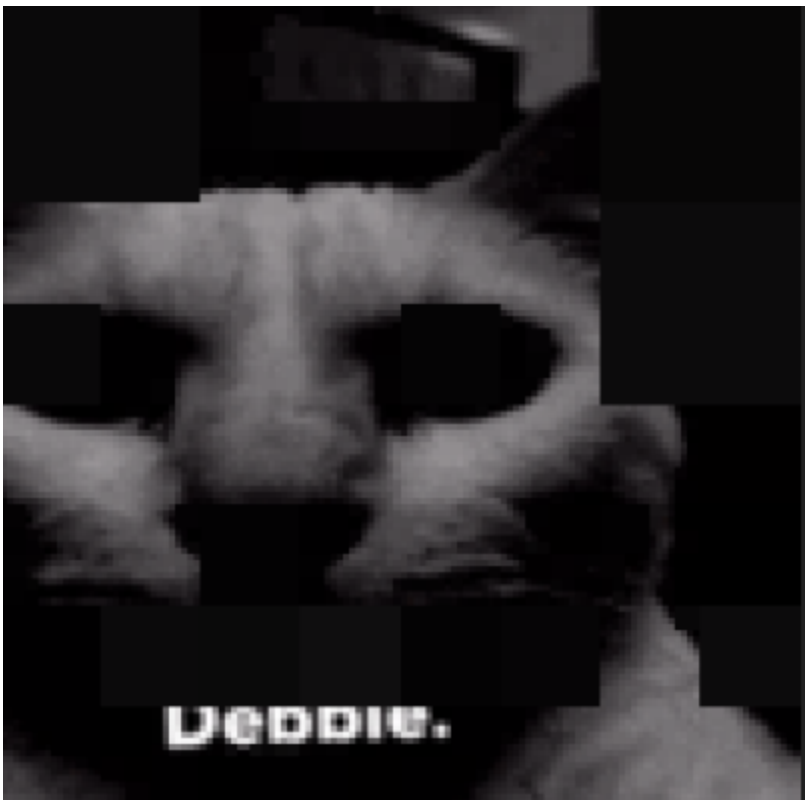
Input Perhitungan Error Entropy:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Debbie.png
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
3
Masukkan ambang batas (sesuai dengan metode yang dipilih): 3
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolEnt.png
```

Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.1192485 detik
Ukuran gambar sebelum: 142221 bytes
Ukuran gambar setelah: 25367 bytes
Persentase kompresi: 82.16367484408069%
Kedalaman pohon: 7
Banyak simpul pada pohon: 7135
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolEnt.png
```

Output Gambar:



3.5 Pengujian 5

Input Gambar png:

Input Perhitungan Error Max Pixel Difference:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Debbie.png
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
2
Masukkan ambang batas (sesuai dengan metode yang dipilih): 20
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolMPD.png
```

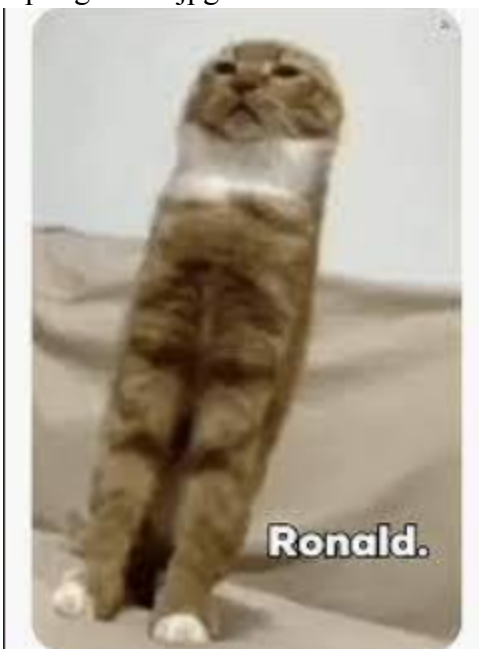
Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.1098721 detik
Ukuran gambar sebelum: 142221 bytes
Ukuran gambar setelah: 25826 bytes
Persentase kompresi: 81.84093769555832%
Kedalaman pohon: 7
Banyak simpul pada pohon: 6175
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\DebbieSolMPD.png
```

Output Gambar:

3.6 Pengujian 6

Input gambar jpg:



Input Perhitungan Error SSIM:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Ronald.jpg
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
5
Masukkan ambang batas (sesuai dengan metode yang dipilih): 0.9
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\RonaldSolSSIM.jpg
```

Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.0888656 detik
Ukuran gambar sebelum: 5327 bytes
Ukuran gambar setelah: 6995 bytes
Persentase kompresi: -31.312183217570855%
Kedalaman pohon: 6
Banyak simpul pada pohon: 3031
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\RonaldSolSSIM.jpg
```

Output Gambar:

3.7 Pengujian 7

Input gambar jpeg:

Input Perhitungan Error SSIM:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Larry.jpeg
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
5
Masukkan ambang batas (sesuai dengan metode yang dipilih): 0.9
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\LarrySolSSIM.jpeg
```

Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.0907959 detik
Ukuran gambar sebelum: 3565 bytes
Ukuran gambar setelah: 5127 bytes
Persentase kompresi: -43.8148667601683%
Kedalaman pohon: 6
Banyak simpul pada pohon: 1438
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\LarrySolSSIM.jpeg
```

Output Gambar:

3.8 Pengujian 8

Input gambar jpeg:

Input perhitungan Error Variance:

```
Masukkan alamat absolut gambar yang akan dikompresi (jpg, png, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\Larry.jpeg
Pilih metode perhitungan error:
1. Mean Absolute Deviation (MAD)
2. Max Pixel Difference
3. Entropy
4. Variance
5. Structural Similarity Index (SSIM)
Masukkan Pilihan (1 - 5):
4
Masukkan ambang batas (sesuai dengan metode yang dipilih): 1000
Masukkan ukuran blok minimum: 4
Masukkan alamat absolut gambar hasil kompresi (png, jpg, atau jpeg): D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\LarrySolVar.jpeg
```

Output:

```
Menyimpan hasil kompresi...
Waktu eksekusi: 0.0806204 detik
Ukuran gambar sebelum: 3565 bytes
Ukuran gambar setelah: 4321 bytes
Persentase kompresi: -21.206171107994386%
Kedalaman pohon: 6
Banyak simpul pada pohon: 469
Gambar hasil kompresi telah disimpan di: D:\CompSCI_ITB\Materi\Semester_4\STIMA\Tucil\Tucil_2\test\LarrySolVar.jpeg
```

Output gambar:

BAB 4 Analisis

1. **Prinsip Algoritma Divide and Conquer**
   Gambar awal dibagi ke dalam empat kuadran/blok secara rekursif. Proses pembagian dilakukan menggunakan fungsi buildQuadTree dari kelas QuadTreeBuilder. Setiap blok dinilai apakah memiliki perbedaan warna yang signifikan berdasarkan metode error tertentu (shouldDivided pada kelas BlockDivision). Jika nilai error dari blok lebih besar dari ambang batas (threshold), maka blok tersebut dibagi kembali menjadi empat sub-blok. Jika tidak, maka blok dianggap homogen dan dinormalisasi warnanya dengan nilai rata-rata RGB seluruh piksel di dalamnya (normalizeColor dari ColorNormalizer). Pendekatan ini memungkinkan area gambar yang seragam (seperti latar belakang polos) hanya direpresentasikan oleh satu node leaf, mengurangi jumlah data secara signifikan
2. **Kualitas Output**

Metode	**Ciri-Ciri**	**Kualitas Output Visual**	**Ukuran Pohon**
Mean Absolute Deviation	Sensitif rata-rata deviasi	Baik untuk gambar halus	Sedang
Max Pixel DIfference	Deteksi selisih tajam	Cepat tapi kasat	Kecil (jika threshold tinggi)
Entropy	Kompleksitas informasi	Tinggi untuk gambar detail	Besar
Variance	Penyebaran warna RGB	Mirip dengan Mean Absolute Deviation tetapi lebih sensitif	Sedang - besar
SSIM	Struktural dan Visual	Paling akurat	Paling besar

Semakin besar pohon (banyak node), semakin tinggi kualitas gambar yang dipertahankan, namun kompresi lebih rendah. Sebaliknya, semakin kecil pohon, kualitas bisa menurun tapi ukuran file hasil lebih kecil.

Semakin kecil threshold, semakin sensitif deteksi blok, menyebabkan lebih banyak node. Threshold besar menghasilkan pohon kecil tapi bisa mengaburkan detail visual. Ukuran minBlockSize digunakan untuk mencegah pembagian hingga skala piksel. Nilai umum: 4 atau 8 piksel. Ini penting untuk menjaga performa dan mencegah overhead tak perlu.

3. **Struktur dan Kompleksitas Pohon**

   Struktur pohon sangat bergantung pada ErrorMethod dan nilai threshold:

   - Jumlah Node: Semakin banyak detail dan semakin kecil threshold, maka node akan lebih banyak karena banyak pembagian. Untuk threshold rendah (misal 10 pada MAD), node bisa mencapai ribuan bahkan puluhan ribu.
   - Kedalaman Maksimum (Max Depth): Menunjukkan seberapa dalam pohon menyelami sub-blok. Biasanya dibatasi oleh minBlockSize. Gambar dengan banyak detail akan memiliki depth lebih besar, yang bisa mencapai $\log_2(n)$, tergantung ukuran gambar dan skala pembagian.
   - Distribusi Node:
     - ❖ Daerah homogen → pohon pendek, node sedikit.
     - ❖ Daerah kompleks (tekstur, noise, tepi) → pohon dalam, node banyak.
   - Implementasi menyimpan node menggunakan kelas QuadTreeNode, yang menyimpan koordinat, dimensi, kedalaman, dan warna rata-rata untuk leaf node.

4. **Kompleksitas Waktu Algoritma**

   - Kasus Terburuk (Worst Case): Ketika setiap piksel memiliki warna unik (misal, gambar noise atau high-detail), algoritma membagi setiap blok hingga ukuran minimum, sehingga kompleksitas menjadi $O(n^2)$, di mana n adalah ukuran sisi gambar (misalnya gambar 512×512 berarti hingga ~262.144 node).
   - Kasus Terbaik (Best Case): Jika gambar sangat homogen, pembagian berhenti pada level awal. Maka kompleksitas waktu bisa turun ke O(1) atau mendekati O(log n), karena hanya beberapa node besar yang perlu dibuat dan diproses.
   - Kompleksitas Nyata (Average Case): Untuk gambar umum (foto, ikon, kartun), kompleksitas berkisar antara O(n log n) hingga $O(n^2)$, tergantung distribusi konten visual dan nilai threshold yang digunakan.

BAB 5 Penjelasan Implementasi Bonus SSIM

```
// SSIM
    public static double ssimRGB(ImageProcessing ori, ImageProcessing comp){
        if (ori.getWidth() != comp.getWidth() || ori.getHeight() != comp.getHeight()){
            throw new IllegalArgumentException("Ukuran blok tidak sama untuk perhitungan
SSIM.");
        }

        double ssimR = ssimChannel(ori, comp, 'R');
        double ssimG = ssimChannel(ori, comp, 'G');
        double ssimB = ssimChannel(ori, comp, 'B');

        double wR = 1.0 / 3.0;
        double wG = 1.0 / 3.0;
        double wB = 1.0 / 3.0;

        return wR * ssimR + wG * ssimG + wB * ssimB;
    }

    private static double ssimChannel(ImageProcessing ori, ImageProcessing comp, char
channel){
        int height = ori.getHeight();
        int width = ori.getWidth();

        int N = width * height;

        double sumO = 0.0, sumC = 0.0;
        double sumO2 = 0.0, sumC2 = 0.0;
        double sumOC = 0.0;

        for (int y = 0; y < height; y++){
            for (int x = 0; x < width; x++){
                Color co = ori.getPixel(y, x);
                Color cc = comp.getPixel(y, x);

                int vo = (channel == 'R') ? co.getRed() :
                (channel == 'G') ? co.getGreen() : co.getBlue();

                int vc = (channel == 'R') ? cc.getRed() :
                (channel == 'G') ? cc.getGreen() : cc.getBlue();

                sumO += vo;
                sumC += vc;
                sumO2 += (vo * (double) vo);
                sumC2 += (vc * (double) vc);
                sumOC += (vo * (double) vc);
            }
        }
```

```
        double meanO = sumO / N;
        double meanC = sumC / N;

        double varO = (sumO2 / N) - (meanO * meanO);
        double varC = (sumC2 / N) - (meanC * meanC);

        double covOC = (sumOC / N) - (meanO * meanC);

        double c1 = (k1 * L) * (k1 * L);
        double c2 = (k2 * L) * (k2 * L);

        double numerator   = (2 * meanO * meanC + c1) * (2 * covOC + c2);
        double denominator = (meanO * meanO + meanC * meanC + c1) * (varO + varC + c2);

        return numerator / denominator;
    }
```

Penjelasan implementasi metode SSIM (Structural Similarity Index Measure):
Metode SSIM adalah metode untuk menilai kualitas gambar yang telah dikompresi dengan membandingkannya dengan gambar aslinya, dengan mempertimbangkan aspek persepsi visual seperti luminasi, kontras, dan struktur.

a. Fungsi ssimChannel menerima parameter gambar asli (ori), gambar yang dibandingkan (comp), dan channel warna yang akan dianalisis.
b. Kode menghitung tiga komponen utama SSIM:
    - Luminansi: Diwakili oleh perhitungan mean dari kedua gambar (meanO dan meanC)
    - Kontras: Diwakili oleh perhitungan variance dari kedua gambar (varO dan varC)
    - Struktur: Diwakili oleh perhitungan covariance antar kedua gambar (covOC)
c. Fungsi ini menghitung akumulasi nilai piksel untuk channel yang dipilih dengan iterasi pada setiap piksel gambar, mengekstrak nilai R, G, atau B sesuai parameter.
d. Konstanta stabilisasi c1 dan c2 dihitung berdasarkan parameter k1, k2, dan L. Konstanta ini penting untuk stabilitas perhitungan ketika variansi atau nilai mean mendekati nol.

BAB 6 Lampiran

Link github : https://github.com/Azzkaaaa/Tucil2_13523125_13523137.git

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	✔	

DAFTAR PUSTAKA

IF2211 Strategi Algoritma - Semester II Tahun 2023/2024
https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-(69).pdf