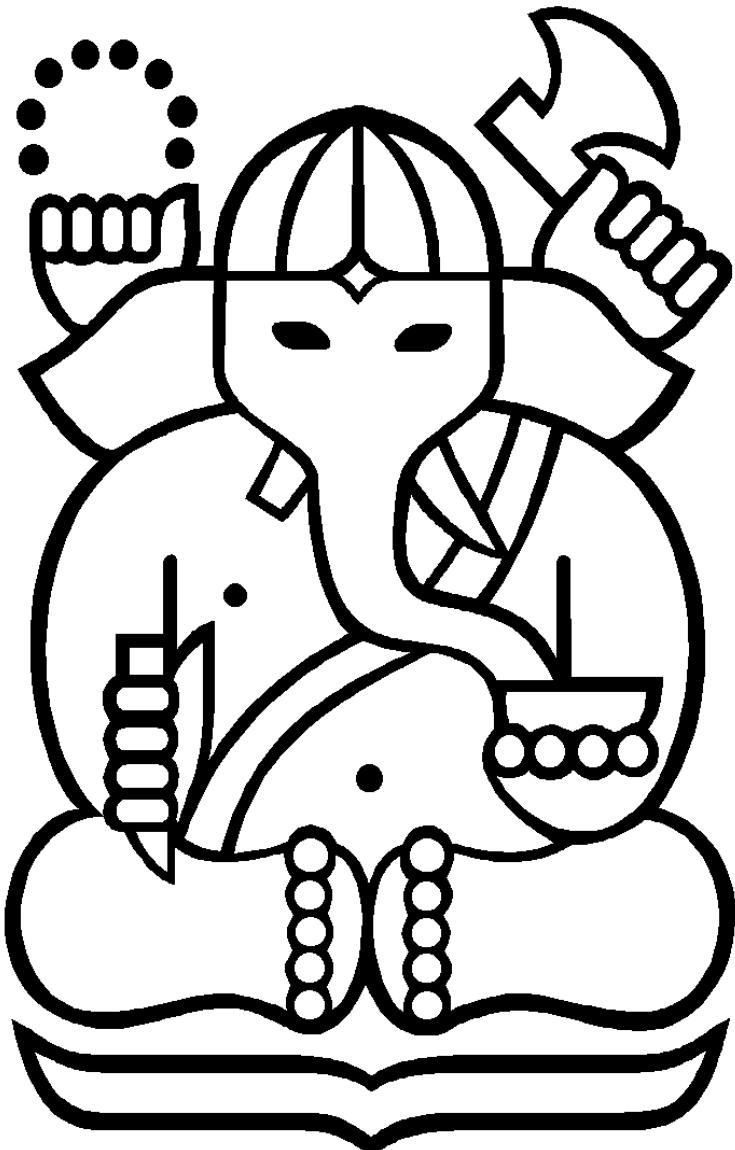


Laporan Tugas Kecil 2

Kompresi Gambar Dengan Metode Quadtree
IF2211 - Strategi Algoritma



Oleh:
Orvin Andika Ikhsan Abhista - 13523017

**Program Studi Teknik Informatika Sekolah Teknik Elektro
dan Informatika Institut Teknologi Bandung 2024**

Daftar Isi

Daftar Isi.....	2
Bab 1.....	3
Analisis Algoritma.....	3
1.1. Algoritma Divide and Conquer.....	3
1.2. Struktur Data Quadtree.....	3
1.3. Implementasi Algoritma Divide and Conquer untuk Kompresi Gambar.....	4
Bab 2.....	5
Source Code Program.....	5
2.1. Source Code Kelas QuadTreeNode.....	5
2.2. Source Code Kelas QuadTree.....	5
2.3. Source Code Kelas Error Calculator.....	8
2.4. Source Code Kelas VarianceRGB.....	8
2.5. Source Code Kelas MeanAbsoluteDeviation.....	9
2.6. Source Code Kelas MaxPixelDifference.....	10
2.7. Source Code Kelas EntropyCalculator.....	11
2.8. Source Code Kelas SSIMCalculator.....	12
2.9. Source Code Kelas TargetThreshold.....	15
2.10. Source Code Main.....	16
Bab 3.....	22
Tangkapan Layar.....	22
3.1. Contoh 1.....	22
3.2. Contoh 2.....	23
3.3. Contoh 3.....	24
3.4. Contoh 4.....	25
3.5. Contoh 5.....	26
3.6. Contoh 6.....	27
3.7. Contoh 7.....	28
Bab 4.....	30
Hasil Analisis.....	30
Bab 5.....	31
Bonus.....	31
5.1. Target Persentase Kompresi.....	31
5.2. Structural Similarity Index.....	31
Lampiran.....	32

Bab 1

Analisis Algoritma

1.1. Algoritma Divide and Conquer

Divide and Conquer adalah algoritma yang bekerja dengan melakukan divide, conquer, dan combine. Divide yang dimaksud adalah membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama). Conquer adalah menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Sementara itu, combine adalah proses menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

1.2. Struktur Data Quadtree

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

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

1.3. Implementasi Algoritma Divide and Conquer untuk Kompresi Gambar

Algoritma Divide and Conquer bisa digunakan untuk melakukan kompresi gambar dengan memanfaatkan struktur data quadtree. Untuk melakukan hal tersebut, pertama-tama program harus membaca gambar input dan membangun struktur quadtree secara rekursif. Setiap node dalam quadtree akan merepresentasikan suatu blok pada gambar dengan posisi, ukuran, dan nilai warna rata-rata jika node tersebut adalah daun.

Saat membangun quadtree, setiap blok dicek apakah layak untuk dikompresi menjadi satu warna berdasarkan metode perhitungan error yang dipilih pengguna. Metode perhitungan error tersebut antara lain, Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy, dan Structural Similarity Index. Variance adalah mengukur seberapa tersebar nilai warna piksel dalam suatu blok. Variance dihitung untuk masing-masing kanal warna (R, G, B), kemudian dirata-ratakan. Mean Absolute Deviation bekerja dengan menghitung rata-rata selisih absolut antara setiap piksel dan warna rata-rata blok tersebut, untuk masing-masing kanal R, G, dan B. Max Pixel Difference bekerja dengan mencari perbedaan maksimal antara warna piksel manapun di dalam blok dengan warna rata-rata blok tersebut. Entropy bekerja dengan mengukur keragaman informasi atau kompleksitas dalam suatu blok, berdasarkan distribusi warna. Sementara itu, Structural Similarity Index adalah metode yang digunakan untuk mengukur kemiripan antara dua gambar berdasarkan struktur visualnya, bukan sekadar per piksel. Pengguna akan memberikan threshold error dan ukuran blok minimum yang akan digunakan program untuk memutuskan apakah blok tersebut harus dibagi atau tidak. Jika nilai error suatu blok lebih kecil dari threshold atau ukurannya sudah mencapai batas minimum blok, maka blok tersebut dianggap seragam dan dijadikan daun pohon, dengan nilai warna rata-rata. Jika tidak, blok akan dibagi menjadi empat sub-blok yang lebih kecil, dan proses ini diulang secara rekursif. Setelah pohon selesai dibangun, program akan merekonstruksi gambar berdasarkan struktur pohon yang dibentuk.

Bab 2

Source Code Program

2.1. Source Code Kelas QuadTreeNode

```
import java.awt.Color;

public class QuadTreeNode {
    int x, y, width, height;
    Color averageColor;
    boolean isLeaf;
    QuadTreeNode[] children;

    public QuadTreeNode(int x, int y, int width, int height)
    {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.isLeaf = false;
        this.children = new QuadTreeNode[4];
    }

    public void setAsLeaf(Color avgColor) {
        this.isLeaf = true;
        this.averageColor = avgColor;
        this.children = null;
    }
}
```

2.2. Source Code Kelas QuadTree

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

```
public class QuadTree {  
    private QuadTreeNode root;  
    private ErrorCalculator calculator;  
    private double threshold;  
    private int minBlockSize;  
    private int nodeCount = 0;  
    private int maxDepth = 0;  
  
    public QuadTree(BufferedImage image, ErrorCalculator  
calculator, double threshold, int minBlockSize) {  
        this.calculator = calculator;  
        this.threshold = threshold;  
        this.minBlockSize = minBlockSize;  
        this.root = build(image, 0, 0, image.getWidth(),  
image.getHeight(), 0);  
    }  
  
    private QuadTreeNode build(BufferedImage image, int x,  
int y, int width, int height, int depth) {  
        nodeCount++;  
        maxDepth = Math.max(maxDepth, depth);  
  
        QuadTreeNode node = new QuadTreeNode(x, y, width,  
height);  
        if (width <= minBlockSize || height <= minBlockSize  
|| calculator.calculateError(image, x, y, width, height) <=  
threshold) {  
            node.setAsLeaf(averageColor(image, x, y, width,  
height));  
            return node;  
        }  
  
        int w2 = width / 2;  
        int h2 = height / 2;  
  
        node.children[0] = build(image, x, y, w2, h2, depth  
+ 1);  
        node.children[1] = build(image, x + w2, y, width -  
w2, h2, depth + 1);  
        node.children[2] = build(image, x, y + h2, w2,  
height - h2, depth + 1);  
    }  
}
```

```

        node.children[3] = build(image, x + w2, y + h2,
width - w2, height - h2, depth + 1);

        return node;
    }

    private Color averageColor(BufferedImage image, int x,
int y, int width, int height) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = width * height;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            sumR += c.getRed();
            sumG += c.getGreen();
            sumB += c.getBlue();
        }
    }
    return new Color((int)(sumR / count), (int)(sumG /
count), (int)(sumB / count));
}

public BufferedImage reconstruct(int width, int height)
{
    BufferedImage result = new BufferedImage(width,
height, BufferedImage.TYPE_INT_RGB);
    fillImage(root, result);
    return result;
}

private void fillImage(QuadTreeNode node, BufferedImage image) {
    if (node.isLeaf) {
        for (int i = node.x; i < node.x + node.width;
i++) {
            for (int j = node.y; j < node.y +
node.height; j++) {
                image.setRGB(i, j,
node.averageColor.getRGB());
            }
        }
    } else {

```

```
        for (QuadTreeNode child : node.children) {
            fillImage(child, image);
        }
    }

    public int getNodeCount() {
        return nodeCount;
    }

    public int getMaxDepth() {
        return maxDepth;
    }
}
```

2.3. Source Code Kelas Error Calculator

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

public interface ErrorCalculator {
    double calculateError(BufferedImage image, int x, int y,
int width, int height);
}
```

2.4. Source Code Kelas VarianceRGB

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

public class VarianceRGB implements ErrorCalculator {
    public double calculateError(BufferedImage image, int x,
int y, int width, int height) {
        long sumR = 0, sumG = 0, sumB = 0;
        int n = width * height;
```

```

        for (int i = x; i < x + width; i++) {
            for (int j = y; j < y + height; j++) {
                Color c = new Color(image.getRGB(i, j));
                sumR += c.getRed();
                sumG += c.getGreen();
                sumB += c.getBlue();
            }
        }

        double meanR = sumR / (double)n;
        double meanG = sumG / (double)n;
        double meanB = sumB / (double)n;

        double variance = 0.0;
        for (int i = x; i < x + width; i++) {
            for (int j = y; j < y + height; j++) {
                Color c = new Color(image.getRGB(i, j));
                variance += Math.pow(c.getRed() - meanR, 2);
                variance += Math.pow(c.getGreen() - meanG,
2);
                variance += Math.pow(c.getBlue() - meanB,
2);
            }
        }

        return variance / (3 * n);
    }
}

```

2.5. Source Code Kelas MeanAbsoluteDeviation

```

import java.awt.Color;
import java.awt.image.BufferedImage;

public class MeanAbsoluteDeviation implements
ErrorCalculator {
    @Override
    public double calculateError(BufferedImage image, int x,

```

```

int y, int width, int height) {
    int count = width * height;
    long sumR = 0, sumG = 0, sumB = 0;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            sumR += c.getRed();
            sumG += c.getGreen();
            sumB += c.getBlue();
        }
    }

    double meanR = sumR / (double) count;
    double meanG = sumG / (double) count;
    double meanB = sumB / (double) count;

    double mad = 0;
    double madR = 0, madG = 0, madB = 0;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            madR += Math.abs(c.getRed() - meanR);
            madG += Math.abs(c.getGreen() - meanG);
            madB += Math.abs(c.getBlue() - meanB);
        }
    }

    madR /= count;
    madG /= count;
    madB /= count;
    mad = madR + madG + madB;
    return mad / (3); // Rata-rata MAD per channel
}
}

```

2.6. Source Code Kelas MaxPixelDifference

```

import java.awt.Color;

```

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

public class MaxPixelDifference implements ErrorCalculator {
    @Override
    public double calculateError(BufferedImage image, int x,
int y, int width, int height) {
        int minR = 255, minG = 255, minB = 255;
        int maxR = 0, maxG = 0, maxB = 0;

        for (int i = x; i < x + width; i++) {
            for (int j = y; j < y + height; j++) {
                Color c = new Color(image.getRGB(i, j));
                int r = c.getRed();
                int g = c.getGreen();
                int b = c.getBlue();
                minR = Math.min(minR, r);
                minG = Math.min(minG, g);
                minB = Math.min(minB, b);
                maxR = Math.max(maxR, r);
                maxG = Math.max(maxG, g);
                maxB = Math.max(maxB, b);
            }
        }

        return (maxR - minR + maxG - minG + maxB - minB) /
3.0;
    }
}
```

2.7. Source Code Kelas EntropyCalculator

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

public class EntropyCalculator implements ErrorCalculator {
    @Override
    public double calculateError(BufferedImage image, int x,
```

```

int y, int width, int height) {
    int[] histogram = new int[256];
    int count = width * height * 3;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            histogram[c.getRed()]++;
            histogram[c.getGreen()]++;
            histogram[c.getBlue()]++;
        }
    }

    double entropy = 0.0;
    for (int freq : histogram) {
        if (freq > 0) {
            double p = freq / (double) count;
            entropy -= p * Math.log(p) / Math.log(2);
        }
    }

    return entropy;
}
}

```

2.8. Source Code Kelas SSIMCalculator

```

import java.awt.Color;
import java.awt.image.BufferedImage;

public class SSIMCalculator implements ErrorCalculator {

    private static final double K1 = 0.01;
    private static final double K2 = 0.03;
    private static final int L = 255;
    private static final double C1 = Math.pow(K1 * L, 2);
    private static final double C2 = Math.pow(K2 * L, 2);
}

```

```
private static final double wR = 0.299;
private static final double wG = 0.587;
private static final double wB = 0.114;

@Override
public double calculateError(BufferedImage image, int x,
int y, int width, int height) {
    Color avg = averageColor(image, x, y, width,
height);

    double ssimR = computeSSIMForChannel(image, x, y,
width, height, avg.getRed(), 'r');
    double ssimG = computeSSIMForChannel(image, x, y,
width, height, avg.getGreen(), 'g');
    double ssimB = computeSSIMForChannel(image, x, y,
width, height, avg.getBlue(), 'b');

    double ssim = wR * ssimR + wG * ssimG + wB * ssimB;
    return 1.0 - ssim;
}

private double computeSSIMForChannel(BufferedImage img,
int x, int y, int w, int h, int avg, char channel) {
    double meanOrig = 0, meanAvg = avg;
    double varOrig = 0, covar = 0;
    int n = w * h;

    int[] values = new int[n];
    int idx = 0;

    for (int i = x; i < x + w; i++) {
        for (int j = y; j < y + h; j++) {
            Color c = new Color(img.getRGB(i, j));
            int val = switch (channel) {
                case 'r' -> c.getRed();
                case 'g' -> c.getGreen();
                case 'b' -> c.getBlue();
                default -> 0;
            };
            values[idx++] = val;
            meanOrig += val;
        }
    }
    meanAvg = meanOrig / n;
    varOrig = covar / n;
    covar = 0;
    for (int i = x; i < x + w; i++) {
        for (int j = y; j < y + h; j++) {
            Color c = new Color(img.getRGB(i, j));
            int val = switch (channel) {
                case 'r' -> c.getRed();
                case 'g' -> c.getGreen();
                case 'b' -> c.getBlue();
                default -> 0;
            };
            values[idx++] = val;
            covar += (val - meanAvg) * (val - meanOrig);
        }
    }
    covar /= n;
    double ssim = (meanAvg - meanOrig) * (meanAvg - meanOrig) +
(covar - (meanAvg * meanOrig)) * (covar - (meanAvg * meanOrig));
    return 1.0 - ssim;
}
```

```
        }

    }

    meanOrig /= n;

    for (int i = 0; i < n; i++) {
        varOrig += Math.pow(values[i] - meanOrig, 2);
        covar += (values[i] - meanOrig) * (meanAvg -
avg);
    }

    varOrig /= n;
    covar /= n;

    double numerator = (2 * meanOrig * meanAvg + C1) *
(2 * covar + C2);
    double denominator = (meanOrig * meanOrig + meanAvg
* meanAvg + C1) * (varOrig + C2);

    return denominator != 0 ? numerator / denominator :
1.0;
}

private Color averageColor(BufferedImage image, int x,
int y, int width, int height) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = width * height;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            sumR += c.getRed();
            sumG += c.getGreen();
            sumB += c.getBlue();
        }
    }

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

2.9. Source Code Kelas TargetThreshold

```
import java.awt.image.BufferedImage;
import java.io.ByteArrayOutputStream;
import javax.imageio.ImageIO;

public class TargetThreshold {
    public static double findOptimalThreshold(
        BufferedImage image,
        ErrorCalculator calculator,
        int minBlockSize,
        double targetCompressionRatio,
        double low,
        double high
    ) {
        if (image == null || calculator == null ||
targetCompressionRatio <= 0.0 || targetCompressionRatio >=
1.0) {
            return -1;
        }

        double bestThreshold = low;
        double bestError = Double.MAX_VALUE;

        long originalSize = getImageSizeInBytes(image,
"png"); // or "jpg" for JPEG
        double epsilon = 0.01;

        while (high - low > epsilon) {
            double mid = (low + high) / 2.0;
            QuadTree qt = new QuadTree(image, calculator,
mid, minBlockSize);
            BufferedImage compressedImage =
qt.reconstruct(image.getWidth(), image.getHeight());

            long compressedSize =
getImageSizeInBytes(compressedImage, "png");
        }
    }
}
```

```
        double compressionRatio = 1.0 - ((double) compressedSize / originalSize);
        double error = Math.abs(compressionRatio - targetCompressionRatio);

        if (error < bestError) {
            bestError = error;
            bestThreshold = mid;
        }

        if (compressionRatio < targetCompressionRatio) {
            low = mid;
        } else {
            high = mid;
        }
    }

    return bestThreshold;
}

public static long getImageSizeInBytes(BufferedImage image, String format) {
    try {
        ByteArrayOutputStream baos = new ByteArrayOutputStream();
        ImageIO.write(image, format, baos);
        return baos.size();
    } catch (Exception e) {
        e.printStackTrace();
        return Long.MAX_VALUE; // Gagal evaluasi
    }
}
}
```

2.10. Source Code Main

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

```
import java.io.File;
import java.util.Scanner;
import javax.imageio.ImageIO;
public class Main {
    public static void main(String[] args) {
        try {
            Scanner scanner = new Scanner(System.in);
            System.out.print("Masukkan alamat absolut gambar input: ");
            String inputPath = scanner.nextLine(); // Alamat Absolut Gambar Input
            System.out.print("Masukkan alamat absolut gambar output: ");
            String outputPath = scanner.nextLine(); // Alamat Absolut Gambar Hasil Kompresi
            System.out.println("Pilih metode perhitungan error: ");
            System.out.println("1. Variance");
            System.out.println("2. Mean Absolute Deviation (MAD)");
            System.out.println("3. Max Pixel Difference (MPD)");
            System.out.println("4. Entropy");
            System.out.println("5. SSIM (Structural Similarity Index)");
            int method; // Metode perhitungan error
            double threshold; // Threshold
            int minBlockSize; // Ukuran minimum blok
            double target; // Target rasio kompresi (0.0 - 1.0)
            while (true) {
                System.out.print("Masukkan pilihan (1-5): ");
                method = scanner.nextInt(); // Metode perhitungan error
                if (method >= 1 && method <= 5) {
                    break; // Valid method selected
                } else {
                    System.out.println("Pilihan tidak valid. Silakan coba lagi.");
                }
            }
        }
    }
}
```

```
        BufferedImage image = ImageIO.read(new
File(inputPath));
        if (image == null) {
            System.out.println("Gambar tidak ditemukan
atau format tidak didukung.");
            scanner.close();
            return; // Gambar tidak ditemukan
        }
        ErrorCalculator errorCalculator; // Perhitungan
error
        double low; // Batas bawah error
        double high; // Batas atas error
        switch (method) {
            case 1 -> {
                errorCalculator = new VarianceRGB();
                low = 0.0;
                high = 65025.0;}
            case 2 -> {
                errorCalculator = new
MeanAbsoluteDeviation();
                low = 0.0;
                high = 255.0;}
            case 3 -> {
                errorCalculator = new
MaxPixelDifference();
                low = 0.0;
                high = 255.0;}
            case 4 -> {
                errorCalculator = new
EntropyCalculator();
                low = 0.0;
                high = 8.0;}
            case 5 -> {
                errorCalculator = new SSIMCalculator();
                low = 0.0;
                high = 1.0;}
            default -> {
                System.out.println("Invalid method
selected.");
                scanner.close();
                return; // Invalid method
            }
        }
```

```
        }

        while (true) {
            System.out.print("Masukkan threshold: ");
            threshold = scanner.nextDouble(); //

            Threshold
                if (method == 5) { // SSIM
                    threshold = 1.0 - threshold; // Invert
                }
                if (threshold >= low && threshold <= high)
                {
                    break; // Valid threshold selected
                } else {
                    System.out.println("Threshold tidak
valid. Silakan coba lagi.");
                }
            }

            while (true) {
                System.out.print("Masukkan ukuran minimum
blok: ");
                minBlockSize = scanner.nextInt(); // Ukuran
minimum blok
                if (minBlockSize > 0 && minBlockSize <=
image.getWidth() && minBlockSize <= image.getHeight()) {
                    break; // Valid block size selected
                } else {
                    System.out.println("Ukuran minimum blok
tidak valid. Silakan coba lagi.");
                }
            }

            while (true) {
                System.out.print("Target persentase
kompresi (0,0 - 1,0) atau 0 untuk default: ");
                target = scanner.nextDouble(); // Target
ratio kompresi (0.0 - 1.0)
                if (target >= 0.0 && target <= 1.0) {
                    break; // Valid target selected
                } else {
                    System.out.println("Target tidak valid.
Silakan coba lagi.");
                }
            }
        }
    }
}
```

```
scanner.close();
if (target > 0) {
    threshold =
TargetThreshold.findOptimalThreshold(image,
errorCalculator, minBlockSize, target, low, high); // Mencari threshold optimal
    if (threshold == -1) {
        System.out.println("Target tidak bisa dicapai.");
        return; // Target not achievable
    }
} else {}
long start = System.currentTimeMillis();
QuadTree qt = new QuadTree(image,
errorCalculator, threshold, minBlockSize);
System.out.println("Processing...");

BufferedImage compressed =
qt.reconstruct(image.getWidth(), image.getHeight());
ImageIO.write(compressed, "png", new
File(outputPath));
long end = System.currentTimeMillis();

System.out.println("Waktu eksekusi: " + (end -
start) + " ms");
System.out.printf("Ukuran Gambar Sebelum: %.2f
KB%n", (TargetThreshold.getImageSizeInBytes(image, "png") /
1000.0));
System.out.printf("Ukuran Gambar Setelah: %.2f
KB%n", (TargetThreshold.getImageSizeInBytes(compressed,
"png") / 1000.0));
System.out.printf("Persentase Kompresi:
%.1f%%\n", (1.0 - ((double)
TargetThreshold.getImageSizeInBytes(compressed, "png") /
TargetThreshold.getImageSizeInBytes(image, "png")) * 100.0);
System.out.println("Kedalaman Pohon: " +
qt.getMaxDepth());
System.out.println("Banyak Simpul: " +
qt.getNodeCount());
```

```
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
```

Bab 3

Tangkapan Layar

3.1. Contoh 1

Input:

```
PS C:\users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output1.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 1
Masukkan threshold: 100
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0
```

Output:

```
Processing...
Waktu eksekusi: 537 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 427,89 KB
Persentase Kompresi: 65,0%
Kedalaman Pohon: 11
Banyak Simpul: 140261
```



3.2. Contoh 2

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output2.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 2
Masukkan threshold: 5
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0
```

Output:

```
Processing...
Waktu eksekusi: 540 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 540,13 KB
Persentase Kompresi: 55,9%
Kedalaman Pohon: 11
Banyak Simpul: 193097
```



3.3. Contoh 3

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output3.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 3
Masukkan threshold: 5
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0
```

Output:

```
Processing...
Waktu eksekusi: 419 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 826,52 KB
Persentase Kompresi: 32,5%
Kedalaman Pohon: 11
Banyak Simpul: 335569
```



3.4. Contoh 4

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output4.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 4
Masukkan threshold: 1
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0
```

Output:

```
Processing...
Waktu eksekusi: 660 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 945,47 KB
Persentase Kompresi: 22,7%
Kedalaman Pohon: 11
Banyak Simpul: 1577301
```



3.5. Contoh 5

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: c:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: c:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output5.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 5
Masukkan threshold: 0,9
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0
```

Output:

```
Processing...
Waktu eksekusi: 1094 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 752,64 KB
Persentase Kompresi: 38,5%
Kedalaman Pohon: 11
Banyak Simpul: 297657
```



3.6. Contoh 6

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output6.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 1
Masukkan threshold: 1
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0,3
```

Output:

```
Processing...
Waktu eksekusi: 411 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 856,44 KB
Persentase Kompresi: 30,0%
Kedalaman Pohon: 11
Banyak Simpul: 359929
```



3.7. Contoh 7

Input:

```
PS C:\Users\Orvin Andika\Downloads\Tucil2_13523017\src> java Main.java
Masukkan alamat absolut gambar input: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\input.png
Masukkan alamat absolut gambar output: C:\Users\Orvin Andika\Downloads\Tucil2_13523017\test\output7.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference (MPD)
4. Entropy
5. SSIM (Structural Similarity Index)
Masukkan pilihan (1-5): 1
Masukkan threshold: 1
Masukkan ukuran minimum blok: 1
Target persentase kompresi (0,0 - 1,0) atau 0 untuk default: 0,7
```

Output:

```
Processing...
Waktu eksekusi: 331 ms
Ukuran Gambar Sebelum: 1223,77 KB
Ukuran Gambar Setelah: 367,14 KB
Persentase Kompresi: 70,0%
Kedalaman Pohon: 11
Banyak Simpul: 114261
```



Bab 4

Hasil Analisis

Kompleksitas waktu program kompresi gambar berbasis Quadtree bergantung pada metode perhitungan error yang digunakan, karena setiap metode memiliki cara yang berbeda dalam mengevaluasi homogenitas blok gambar. Untuk metode Variance (RGB), program menghitung rata-rata dan variansi warna dari setiap kanal (R, G, dan B) dalam sebuah blok. Proses ini memerlukan traversal seluruh piksel dalam blok, sehingga kompleksitasnya sebesar $O(w \times h)$ untuk satu blok, dan secara keseluruhan mencapai $O(W \times H)$ untuk seluruh gambar karena semua piksel dianalisis secara rekursif.

Metode Mean Absolute Deviation (MAD) juga melakukan iterasi ke seluruh piksel dalam blok untuk menghitung deviasi absolut terhadap rata-rata setiap kanal warna. Karena pendekatannya serupa dengan variansi, kompleksitasnya tetap $O(W \times H)$.

Metode Max Pixel Difference (MPD) mencari selisih maksimum antar piksel dalam blok. Meskipun tidak memerlukan rata-rata, ia tetap membandingkan semua pasang piksel dalam blok, menghasilkan kompleksitas $O(w \times h)$ per blok dan $O(W \times H)$ secara total. Namun, dalam implementasi umum (dengan satu loop), MPD sering hanya mencari nilai minimum dan maksimum dalam satu traversal, sehingga kompleksitas efektif per blok tetap $O(w \times h)$.

Untuk metode Entropy, program menghitung histogram kemunculan warna dalam blok, lalu menghitung nilai entropinya berdasarkan distribusi probabilitas tiap nilai kanal. Proses ini juga membutuhkan satu kali traversal setiap piksel dalam blok untuk mengisi histogram, sehingga kompleksitasnya $O(w \times h)$ per blok dan $O(W \times H)$ untuk seluruh gambar.

Metode Structural Similarity Index lebih kompleks karena selain menghitung rata-rata dan variansi seperti variansi biasa, ia juga menghitung kovarians antar piksel dengan nilai rata-rata blok. Perhitungan ini dilakukan secara terpisah untuk tiga kanal (R, G, dan B), sehingga setiap blok diproses tiga kali lebih banyak. Dengan demikian, kompleksitasnya tetap $O(w \times h)$ per blok, tetapi dengan konstanta yang lebih besar. Secara total, kompleksitasnya tetap $O(W \times H)$ namun dengan beban komputasi paling tinggi dibanding metode lain.

Secara umum, karena Quadtree membagi gambar secara rekursif dan semua metode memproses piksel satu per satu dalam blok, kompleksitas waktu keseluruhan untuk setiap metode tetap linear terhadap ukuran gambar ($O(W \times H)$), namun efisiensi aktual sangat dipengaruhi oleh metode yang dipilih dan distribusi warna dalam gambar.

Bab 5

Bonus

5.1. Target Persentase Kompresi

Fitur ini memungkinkan pengguna untuk memasukkan target kompresi yang dikehendaki (0 - 1). Program akan mencari threshold paling optimal untuk menghasilkan persentase kompresi yang diinginkan pengguna. Proses ini dilakukan dengan melakukan binary search terhadap rentang nilai error yang valid untuk metode yang dipilih, misalnya 0-255 untuk MAD dan MPD atau 0-8 untuk Entropy. Program secara iteratif membentuk pohon Quadtree dengan berbagai nilai threshold, kemudian mengukur rasio kompresinya dan membandingkannya dengan target. Jika kompresi terlalu tinggi atau rendah, nilai threshold akan disesuaikan hingga dicapai nilai optimal atau batas iterasi habis. Hal ini memungkinkan program secara otomatis menyesuaikan tingkat kompresi berdasarkan kebutuhan pengguna, tanpa harus menebak nilai threshold secara manual.

5.2. Structural Similarity Index

Structural Similarity Index (SSIM) adalah etode perhitungan error yang menilai kualitas blok gambar dengan mempertimbangkan persepsi manusia terhadap struktur dan tekstur. Tidak seperti metode lain yang hanya membandingkan perbedaan nilai warna secara langsung, SSIM mempertimbangkan tiga komponen utama: luminansi (tingkat kecerahan), kontras, dan struktur. Dalam implementasi program, SSIM dihitung untuk setiap kanal warna (merah, hijau, biru), lalu dirata-rata berdasarkan bobot persepsi warna manusia. Nilai SSIM berkisar dari 0 hingga 1, di mana 1 berarti blok sangat mirip dengan rata-rata, dan semakin kecil nilai SSIM, semakin besar perbedaan strukturnya. Karena SSIM mempertimbangkan konteks spasial dan hubungan antar piksel, metode ini cenderung menghasilkan kualitas visual yang lebih baik, meskipun lebih kompleks secara komputasi.

Lampiran

Pranala Repository: https://github.com/orvin14/Tucil2_13523017

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	✓	