

TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Kompresi Gambar dengan Quadtree (*Divide and Conquer*)



Disusun oleh :

Jonathan Levi	13523132
Jovandra Otniel Pangalambok Siregar	13523141

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2023

Daftar Isi

BAGIAN I - Quadtree.....	3
1.1 Pendahuluan.....	3
1.2 Konsep Dasar Quadtree.....	3
1.3 Mekanisme Kerja Quadtree.....	3
1.4 Kelebihan dan Kekurangan Quadtree.....	5
1.5 Aplikasi Quadtree.....	5
BAGIAN II - Source Code.....	6
2.1 File Main.java.....	7
2.2 File CompressConfig.java.....	14
2.3 File ErrorCalc.java.....	14
2.4 File VarErrorCalc.java.....	15
2.5 File MADErrorCalc.java.....	16
2.6 File PixelDiffErrorCalc.java.....	17
2.7 File EntropyErrorCalc.java.....	18
2.8 File SSIMErrorCalc.java.....	19
2.9 File QuadTree.java.....	21
2.10 File QuadTreeNode.java.....	29
2.11 File GIFSeqWriter.java.....	30
BAGIAN III - Test Case.....	32
BAGIAN IV - Analisis Algoritma Divide and Conquer.....	37
BAGIAN V - Bonus.....	39
BAGIAN VI - Lampiran.....	41

BAGIAN I

Quadtree

Quadtree adalah algoritma *divide and conquer* yang dipakai dalam kompresi gambar.

1.1 Pendahuluan

Pada era digital saat ini, pengolahan data (terutama data gambar) memerlukan teknik efisien dalam penyimpanan dan proses (supaya cepat dan hemat ruang). Salah satu teknik tersebut adalah Quadtree (dibaca: "kwad-tree"), sebuah struktur data hierarkis yang sangat berguna untuk membagi ruang atau data menjadi empat bagian secara rekursif. Quadtree (sering juga disebut "quad-tree") digunakan di berbagai aplikasi, mulai dari pengolahan citra (image processing) hingga pemetaan dan game (contoh: collision detection).

1.2 Konsep Dasar Quadtree

Quadtree adalah struktur data berbasis pohon, di mana setiap node (simpul) memiliki beberapa anak hingga empat buah anak. Setiap node mewakili suatu area (region) pada ruang dua dimensi. (Secara sederhana, anggap saja bahwa setiap node menyimpan informasi tentang sebuah wilayah pada gambar atau peta.) Quadtree memiliki dua jenis utama node, yaitu:

- **Node Internal:** Node yang tidak memiliki data final (leaf) tetapi memiliki empat sub-blok (child nodes) yang membagi wilayahnya.
- **Node Daun (Leaf Node):** Node yang tidak dibagi lagi (tidak memiliki anak) karena sudah memenuhi kriteria keseragaman atau ukuran minimum.

Sebagai contoh, bayangkan sebuah gambar (misalnya, foto pemandangan). Quadtree (secara rekursif) akan memeriksa (analisis) bagian-bagian gambar tersebut: jika suatu blok gambar homogen (misalnya, warnanya seragam), maka blok itu akan disimpan sebagai daun; namun, jika blok tersebut memiliki banyak variasi warna, maka blok itu akan dibagi menjadi empat sub-blok, dan proses ini akan terus berlanjut sampai kondisi tertentu tercapai (contoh: ukuran blok sudah mencapai batas minimum atau tingkat variasinya sudah cukup rendah).

1.3 Mekanisme Kerja Quadtree

Mekanisme kerja Quadtree dimulai dengan pembagian awal suatu gambar atau data menjadi empat kuadran yang sama besar; (proses ini menyerupai pembagian sebuah persegi ke dalam empat persegi kecil). Setelah pembagian, setiap kuadran dianalisis secara mendalam untuk menentukan tingkat keseragaman atau homogenitas, misalnya, dengan mengukur variansi atau deviasi rata-rata nilai piksel. Jika perbedaan nilai (error) di suatu kuadran berada di bawah ambang tertentu (threshold), wilayah itu dianggap homogen dan tidak perlu dibagi

lagi, sehingga disimpan sebagai node daun; namun, apabila error melebihi threshold dan ukuran wilayah masih mencukupi, kuadran tersebut akan dibagi kembali, dan proses tersebut berlanjut secara rekursif hingga seluruh wilayah mencapai batas homogenitas atau ukuran minimum yang ditentukan.

Berikut adalah *pseudocode* dari algoritma *divide and conquer* yang digunakan:

```

FUNCTION BuildTreeRecursive(x, y, width, height)
    { Pastikan koordinat dan ukuran berada dalam batas image }
    IF (x<0 OR y<0 OR x+width > image.width OR y+height >
    image.height) THEN
        x ← MAX(0, x)
        y ← MAX(0, y)
        width ← MIN(width, image.width - x)
        height ← MIN(height, image.height - y)
    ENDIF

    IF (width <= 0 OR height <= 0) THEN
        RETURN null
    ENDIF

    avgColor ← ComputeAverageColor(x, y, width, height)
    error ← errorCalculator.ComputeError(image, x, y, width,
    height, avgColor)

    tooSmall ← (width <= minBlockSize OR height <= minBlockSize)
    cannotSplitFurther ← (width <= 1 OR height <= 1)
    errorAcceptable ← (error <= threshold)

    { Kondisi daun: Error sudah acceptable atau ukuran tidak
    memungkinkan pembelahan }
    IF (errorAcceptable OR tooSmall OR cannotSplitFurther) THEN
        RETURN NewNode(x, y, width, height, avgColor, error)
    ENDIF

    { Membuat node internal dan membagi menjadi empat sub-blok }
    node ← NewNode(x, y, width, height, avgColor, error)
    node.children ← array OF 4

    halfWidth ← floor(width / 2)
    halfHeight ← floor(height / 2)
    width2 ← width - halfWidth { Menangani nilai ganjil }
    height2 ← height - halfHeight { Menangani nilai ganjil }

    node.children[0] ← BuildTreeRecursive(x, y, halfWidth,
    halfHeight)
    node.children[1] ← BuildTreeRecursive(x + halfWidth, y,
    width2, halfHeight)

```

```

node.children[2] ← BuildTreeRecursive(x, y + halfHeight,
halfWidth, height2)
node.children[3] ← BuildTreeRecursive(x + halfWidth, y +
halfHeight, width2, height2)

RETURN node
END FUNCTION

```

1.4 Kelebihan dan Kekurangan Quadtree

Quadtree memiliki berbagai keunggulan, antara lain: (i) *Efisiensi penyimpanan*: dengan merepresentasikan area yang homogen menggunakan satu simpul, redundansi data dapat diminimalkan; (ii) *Fleksibilitas*: struktur hierarkis memudahkan adaptasi subdivisi berdasarkan kompleksitas lokal, sehingga sangat berguna dalam aplikasi yang memerlukan pencarian data spasial; (iii) *Kecepatan akses*: pencarian data dalam struktur Quadtree dapat dilakukan dengan cepat, karena setiap node secara eksplisit mewakili suatu wilayah tertentu.

Di sisi lain, terdapat beberapa kekurangan, seperti: (i) *Kompleksitas rekursif*: proses pembagian yang berlangsung terus-menerus dapat menghasilkan struktur pohon yang sangat dalam dan kompleks; (ii) *Ukuran file output*: dalam beberapa kasus, terutama jika output disimpan dalam format lossless (seperti PNG), hasil kompresi mungkin justru menghasilkan file yang lebih besar karena struktur subdivisi yang banyak; (iii) *Sensitivitas parameter*: penentuan threshold dan ukuran blok minimum merupakan faktor krusial; pilihan parameter yang kurang tepat dapat menghasilkan kompresi yang tidak optimal, baik dari segi kualitas visual maupun ukuran file.

1.5 Aplikasi Quadtree

Quadtree telah diterapkan dalam berbagai aplikasi; contohnya, dalam pengolahan gambar, metode ini digunakan untuk melakukan kompresi (*lossy compression*) dengan cara menghilangkan detail yang tidak penting, sehingga file yang dihasilkan memiliki ukuran yang lebih kecil meskipun kualitas visualnya berkurang. Demikian pula, dalam sistem informasi geografis (GIS), Quadtree memungkinkan kita melakukan pengindeksan dan pencarian data spasial secara efisien; hal ini juga berlaku pada bidang pengembangan game, di mana Quadtree membantu dalam deteksi tabrakan (*collision detection*) dan pengelolaan area permainan secara terstruktur.

BAGIAN II

Source Code

Program ini ditulis menggunakan bahasa pemrograman Java, dengan menggunakan *library* berikut:

- java.io
- java.util
- java.awt
- javax.imageio

2.1 File Main.java

File Main.java

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

public class Main {
    public static void main(String[] args) {
        try (Scanner sc = new Scanner(System.in)) {
            CompressConfig config = new CompressConfig();
            // Input path gambar
            System.out.print("Masukkan lokasi absolut gambar input: ");
            config.inputImagePath = sc.nextLine();
            // Input metode perhitungan error
            System.out.println("Pilih metode perhitungan error:");
            System.out.println("1. Variance");
            System.out.println("2. Mean Absolute Deviation");
            System.out.println("3. Max Pixel Difference");
            System.out.println("4. Entropy");
            System.out.println("5. Structural Similarity Index (SSIM)");
            System.out.print("Masukkan pilihan (angka): ");
            config.errorMethod = sc.nextInt();
            // Input threshold awal
            System.out.print("Masukkan nilai threshold awal: ");
            config.threshold = sc.nextDouble();
            // Input minBlockSize
            System.out.print("Masukkan ukuran blok minimum: ");
            config.minBlockSize = sc.nextInt();
            // Input target compression
            System.out.print("Masukkan target persentase kompresi (floating
number, 1,0 = 100%, 0 utk nonaktif): ");
            config.targetCompression = sc.nextDouble();
            sc.nextLine(); // Buang newline
            // Output path
```

```

        System.out.print("Masukkan lokasi absolut untuk gambar keluaran:
");

        config.outputImagePath = sc.nextLine();

        // GIF options
        System.out.print("Buat GIF visualisasi? (y/n): ");
        config.createGif = sc.nextLine().trim().equalsIgnoreCase("y");

        if (config.createGif) {
            // GIF path
            System.out.print("Masukkan lokasi absolut untuk GIF
visualisasi: ");
            config.gifOutputPath = sc.nextLine();
            // Delay frame GIF
            System.out.print("Masukkan delay antar frame (ms): ");
            config.gifDelay = sc.nextInt();
            // Max frames
            System.out.print("Maksimum jumlah frame (batasi untuk
menghindari OutOfMemory, rekomendasi 50-200): ");
            config.maxGifFrames = sc.nextInt();
            // Frame skip
            System.out.print("Rekam setiap berapa frame (1=semua, 2=setiap
2 frame, dll): ");
            config.gifFrameSkip = sc.nextInt();
            sc.nextLine(); // Buang newline
        }

        // Baca inputImage
        BufferedImage inputImage;
        try {
            File f = new File(config.inputImagePath);
            if (!f.exists()) {
                System.err.println("File tidak ditemukan: " +
config.inputImagePath);
                return;
            }
            inputImage = ImageIO.read(f);
            if (inputImage == null) {
                System.err.println("Format gambar tidak didukung!");
                return;
            }
        }
    }
}

```

```

        // Berikan peringatan untuk gambar besar
        int megapixels = (inputImage.getWidth() *
inputImage.getHeight()) / 1000000;
        if (megapixels > 2) {
            System.out.println("PERINGATAN: Gambar berukuran besar (" +
+ megapixels + " megapixel).");
            if (config.createGif) {
                System.out.println("Pembuatan GIF mungkin akan memakan
banyak memori. Menyesuaikan parameter...");
                config.maxGifFrames = Math.min(config.maxGifFrames,
50);
                config.gifFrameSkip = Math.max(config.gifFrameSkip,
2);
                System.out.println("Disesuaikan: maxFrames=" +
config.maxGifFrames + ", frameSkip=" + config.gifFrameSkip);
            }
        }

    } catch (IOException e) {
        System.err.println("Gagal membaca gambar: " + e.getMessage());
        return;
    }
    // Tentukan ErrorCalc
    ErrorCalc errorCalc;
    switch(config.errorMethod) {
        case 1 -> errorCalc = new VarErrorCalc();
        case 2 -> errorCalc = new MADErrorCalc();
        case 3 -> errorCalc = new PixelDiffErrorCalc();
        case 4 -> errorCalc = new EntropyErrorCalc();
        case 5 -> errorCalc = new SSIMErrorCalc();
        default -> {
            System.err.println("Metode error tidak diimplementasikan,
default ke Variance!");
            errorCalc = new VarErrorCalc();
        }
    }
    // Mulai timer
    long startTime = System.nanoTime();
    // Jika targetCompression > 0, tuning threshold
    if (config.targetCompression > 0) {
        double desiredRatio = config.targetCompression; // 0..1
        double low = 0.0;

```

```

        double high = (config.errorMethod == 1) ? 65025 : 1.0;
        // (Atur rentang high sesuai metode, misal SSIM=1.0,
Var=65025, dsb.)

        double bestThreshold = config.threshold;
        double tolerance = 0.01;
        int maxIter = 20;
        for (int i = 0; i < maxIter; i++) {
            double mid = (low + high) / 2.0;
            // Bangun Quadtree sementara - tanpa GIF recording untuk
tuning
            QuadTree qt = new QuadTree(inputImage, mid,
config.minBlockSize, errorCalc, false, 0, 1);
            qt.buildTree();
            int leaves = qt.countLeaves();
            int totalPixels = inputImage.getWidth() *
inputImage.getHeight();
            double currentCompressionRatio = 1.0 - ( (double) leaves /
totalPixels );

            if (Math.abs(currentCompressionRatio - desiredRatio) <=
tolerance) {
                bestThreshold = mid;
                break;
            }
            if (currentCompressionRatio < desiredRatio) {
                // kompresinya masih kurang (terlalu banyak leaf) =>
turunkan threshold
                high = mid;
            } else {
                // kompresinya kebanyakan => naikin threshold
                low = mid;
            }
            bestThreshold = mid;
        }
        config.threshold = bestThreshold;
        System.out.println("Optimal threshold ditemukan: " +
config.threshold);
    }

    // Bangun Quadtree final dengan pengaturan GIF yang sesuai
    QuadTree quadtree = new QuadTree(

```

```

        inputImage,
        config.threshold,
        config.minBlockSize,
        errorCalc,
        config.createGif,
        config.maxGifFrames,
        config.gifFrameSkip
    );

System.out.println("Membangun quadtree...");
try {
    quadtree.buildTree();
} catch (OutOfMemoryError e) {
    System.err.println("ERROR: Out of memory saat memproses
gambar!");
    System.err.println("Coba lagi dengan parameter berikut:");
    System.err.println("- Nonaktifkan pembuatan GIF");
    System.err.println("- Kurangi ukuran gambar input");
    System.err.println("- Tingkatkan nilai threshold");
    System.err.println("- Tingkatkan ukuran blok minimum");
    return;
}

// Hasil kompresi
System.out.println("Membuat gambar hasil kompresi...");
BufferedImage outputImage = quadtree.generateCompressedImage();
long endTime = System.nanoTime();
double elapsedSec = (endTime - startTime) / 1e9;

// Simpan
try {
    File outputFile = new File(config.outputImagePath);
    // Buat direktori jika belum ada
    File parentDir = outputFile.getParentFile();
    if (parentDir != null && !parentDir.exists()) {
        parentDir.mkdirs();
    }

    // Pilih format output berdasarkan ekstensi file
    String extension =
config.outputImagePath.substring(config.outputImagePath.lastIndexOf('.') + 1);
    if (extension.isEmpty()) extension = "jpg";
}

```

```

        ImageIO.write(outputImage, extension, outputFile);
        System.out.println("Gambar hasil kompresi berhasil disimpan
di: " + config.outputImagePath);
    } catch(IOException e) {
        System.err.println("Gagal menulis gambar: " + e.getMessage());
    }

    // Hitung rasio kompresi
    File inFile = new File(config.inputImagePath);
    File outFile = new File(config.outputImagePath);
    long sizeBefore = inFile.length();
    long sizeAfter = outFile.length();
    double percent = (1.0 - ((double) sizeAfter / sizeBefore)) * 100.0;
    int depth = quadtree.getTreeDepth();
    int leaves = quadtree.countLeaves();

    System.out.println("\n===== Hasil Kompresi =====");
    System.out.printf("Waktu eksekusi : %.3f detik\n",
elapsedSec);
    System.out.printf("Ukuran gambar sebelum: %d bytes\n",
sizeBefore);
    System.out.printf("Ukuran gambar sesudah: %d bytes\n", sizeAfter);
    System.out.printf("Persentase kompresi : %.2f%%\n", percent);
    System.out.println("Kedalaman pohon : " + depth);
    System.out.println("Banyak simpul (leaf) : " + leaves);

    // Buat GIF animasi jika diaktifkan
    if (config.createGif) {
        List<BufferedImage> frames = quadtree.getGifFrames();
        System.out.println("Terekam " + frames.size() + " frame untuk
animasi GIF.");
        if (!frames.isEmpty()) {
            try {
                File gifFile = new File(config.gifOutputPath);
                File gifParentDir = gifFile.getParentFile();
                if (gifParentDir != null && !gifParentDir.exists())
gifParentDir.mkdirs();
                try (ImageOutputStream ios = new
FileImageOutputStream(gifFile)) {

```

```
        GIFSeqWriter gifWriter = new GIFSeqWriter(ios,
BufferedImage.TYPE_INT_RGB, config.gifDelay, true);
        for (BufferedImage frame : frames) {
            gifWriter.writeToSequence(frame);
        }
        gifWriter.close();
    }
    System.out.println("GIF proses kompresi berhasil
disimpan di: " + config.gifOutputPath);
} catch (IOException e) {
    System.err.println("Gagal menulis GIF: " +
e.getMessage());
}
} else {
    System.out.println("Tidak ada frame GIF yang direkam.");
}
}
}
}
```

2.2 File CompressConfig.java

```
public class CompressConfig {
    public String inputImagePath;
    public String outputImagePath;
    public String gifOutputPath;
    public int errorMethod;
    public double threshold;
    public int minBlockSize;
    public double targetCompression; // 0 berarti nonaktif, kalo aktif:
    nilai antara 0 dan 1 (1.0 = 100%)
    public int gifDelay; // delay per frame GIF (ms)
}
```

2.3 File ErrorCalc.java

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

You, 3 days ago | 2 authors (You and one other)
public interface ErrorCalc {
    /**
     * Menghitung error untuk blok gambar [x,y,width,height]
     * berdasarkan nilai rata-rata.
     */
    double computeError(BufferedImage image, int x, int y,
                       int width, int height, Color avgColor);
}
```

2.4 File VarErrorCalc.java

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

You, 54 seconds ago | 2 authors (jovan196 and one other)
public class VarErrorCalc implements ErrorCalc {
    @Override
    public double computeError(BufferedImage image, int x, int y,
        | | | | | | |   int width, int height, Color avgColor) {
        double sumVariance = 0;
        int count = 0;

        // Pastikan tidak melewati ukuran gambar
        int endX = Math.min(x + width, image.getWidth());
        int endY = Math.min(y + height, image.getHeight());
        x = Math.max(0, x);
        y = Math.max(0, y);

        for (int j = y; j < endY; j++) {
            for (int i = x; i < endX; i++) {
                Color c = new Color(image.getRGB(i, j));
                double diffR = c.getRed() - avgColor.getRed();
                double diffG = c.getGreen() - avgColor.getGreen();
                double diffB = c.getBlue() - avgColor.getBlue();
                sumVariance += (diffR*diffR + diffG*diffG + diffB*diffB);
                count++;
            }
        }

        if (count == 0) return 0.0; // Jika dibagi 0

        // Rata-rata variansi tiap kanal (dibagi 3)
        return (sumVariance / count) / 3.0;
    }
}
```

2.5 File MADErrorCalc.java

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

...
public class MADErrorCalc implements ErrorCalc {
    @Override
    public double computeError(BufferedImage image, int x, int y,
        | | | | | | | | int width, int height, Color avgColor) {
        double sumR=0, sumG=0, sumB=0;
        int count = 0;
        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                Color c = new Color(image.getRGB(i, j));
                sumR += Math.abs(c.getRed() - avgColor.getRed());
                sumG += Math.abs(c.getGreen() - avgColor.getGreen());
                sumB += Math.abs(c.getBlue() - avgColor.getBlue());
                count++;
            }
        }
        if (count == 0) count = 1;
        double madR = sumR/count;
        double madG = sumG/count;
        double madB = sumB/count;
        // Rata-rata MAD tiap kanal (dibagi 3)
        return (madR + madG + madB) / 3.0;
    }
}
```

2.6 File PixelDiffErrorCalc.java

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

...
public class PixelDiffErrorCalc implements ErrorCalc {
    @Override
    public double computeError(BufferedImage image, int x, int y,
        | | | | | | | | int width, int height, Color avgColor) {
        int maxR = 0, maxG = 0, maxB = 0;
        int minR = 255, minG = 255, minB = 255;

        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();
                maxR = Math.max(maxR, r);
                maxG = Math.max(maxG, g);
                maxB = Math.max(maxB, b);
                minR = Math.min(minR, r);
                minG = Math.min(minG, g);
                minB = Math.min(minB, b);
            }
        }

        double dR = maxR - minR;
        double dG = maxG - minG;
        double dB = maxB - minB;
        return (dR+dG+dB) / 3;
    }
}
```

2.7 File EntropyErrorCalc.java

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

You, 1 second ago | 1 author (You)
public class EntropyErrorCalc implements ErrorCalc {
    @Override
    public double computeError(BufferedImage image, int x, int y,
        int width, int height, Color avgColor) {
        int[] uniquePxCountR = new int[256];
        int[] uniquePxCountG = new int[256];
        int[] uniquePxCountB = new int[256];
        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));
                uniquePxCountR[c.getRed()]++;
                uniquePxCountG[c.getGreen()]++;
                uniquePxCountB[c.getBlue()]++;
            }
        }

        double hR=0, hG=0, hB=0;

        for (int i=0; i<256; i++) {
            if (uniquePxCountR[i]>0) {
                double pR = uniquePxCountR[i]/count;
                hR -= pR * (Math.Log(pR)/Math.Log(2));
            }
            if (uniquePxCountG[i]>0) {
                double pG = uniquePxCountG[i]/count;
                hG -= pG * (Math.Log(pG)/Math.Log(2));
            }
            if (uniquePxCountB[i]>0) {
                double pB = uniquePxCountB[i]/count;
                hB -= pB * (Math.Log(pB)/Math.Log(2));
            }
        }

        return (hR+hG+hB) / 3;
    }
}
```

2.8 File SSIMErrorCalc.java

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

You, 4 seconds ago | 2 authors (jovan196 and one other)
public class SSIMErrorCalc implements ErrorCalc {
    // Konstanta untuk SSIM gambar 8-bit (K1=0.01, K2=0.03, L=255)
    private final double C1 = 6.5025;
    private final double C2 = 58.5225;

    @Override
    public double computeError(BufferedImage image, int x, int y,
        int width, int height, Color avgColor) {
        // Hitung SSIM per channel dengan asumsi blok konstanta
        double ssimR = computeSSIM(image, x, y, width, height,
            channel:'R', avgColor.getRed());
        double ssimG = computeSSIM(image, x, y, width, height,
            channel:'G', avgColor.getGreen());
        double ssimB = computeSSIM(image, x, y, width, height,
            channel:'B', avgColor.getBlue());
        double avgSSIM = (ssimR + ssimG + ssimB) / 3.0;

        // Konversi ke error (SSIM makin tinggi berarti error makin rendah)
        return 1.0 - avgSSIM;
    }

    private double computeSSIM(BufferedImage image, int x, int y,
        int width, int height,
        char channel, int avgVal) {
        double sum = 0, sumSq = 0, covar = 0;
        int count = 0;

        // Pastikan tidak lewat ukuran gambar
        int endX = Math.min(x + width, image.getWidth());
        int endY = Math.min(y + height, image.getHeight());
        x = Math.max(0, x);
        y = Math.max(0, y);
```

```

// Rata-rata dan variansi original block
for (int j = y; j < endY; j++) {
    for (int i = x; i < endX; i++) {
        Color c = new Color(image.getRGB(i, j));
        int val = 0;
        switch (channel) {
            case 'R' -> val = c.getRed();
            case 'G' -> val = c.getGreen();
            case 'B' -> val = c.getBlue();
            default -> {
            }
        }
        sum += val;
        sumSq += val * val;
        covar += val * avgVal; // Kovariansi dengan blok konstan
        count++;
    }
}

if (count == 0) return 1.0; // Return perfect match, cegah dibagi 0

double mean = sum / count;
double variance = sumSq / count - mean * mean;
double covariance = covar / count - mean * avgVal;
if (variance < 1e-3) variance = 1e-3;

// Hitung SSIM index
double numerator = (2 * mean * avgVal + C1) * (2 * covariance + C2);
double denominator = (mean * mean + avgVal * avgVal + C1) * (variance + C2);

return numerator / denominator;
}
}

```

2.9 File QuadTree.java

```
import java.awt.Color;
import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;

public class QuadTree {

    private BufferedImage image;
    private int minBlockSize;
    private double threshold;
    private ErrorCalc errorCalculator;

    private QuadTreeNode root;
    private List<BufferedImage> gifFrames; // Menyimpan frame GIF (bonus)

    private boolean recordGif = true;
    private int maxFrames = 100; // Jumlah maksimum frame yang akan direkam
    private int frameSkip = 1; // Rekam setiap frame ke-N (1 = rekam semua)
    private int frameCount = 0; // Penghitung untuk melacak frame yang
dilewati
    private int totalFramesPossible = 0; // Penghitung untuk semua frame yang
mungkin

    public QuadTree(BufferedImage image, double threshold, int minBlockSize,
ErrorCalc errorCalculator) {
        this(image, threshold, minBlockSize, errorCalculator, true, 100, 1);
    }

    public QuadTree(BufferedImage image, double threshold, int minBlockSize,
ErrorCalc errorCalculator,
            boolean recordGif, int maxFrames, int frameSkip) {
        this.image = image;
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.errorCalculator = errorCalculator;
        this.gifFrames = new ArrayList<>();
        this.recordGif = recordGif;
        this.maxFrames = maxFrames;
    }
}
```

```

        this.frameSkip = frameSkip;

        // Menonaktifkan perekaman GIF berukuran besar untuk menghindari
masalah memori
        if (recordGif && (image.getWidth() * image.getHeight() > 2000000)) {
// > ~2 megapixels
            System.out.println("Peringatan: Gambar besar terdeteksi. Perekaman
GIF dibatasi hingga " + maxFrames + " frame.");
            this.frameSkip = Math.max(2, frameSkip); // Skip frame untuk
gambar berukuran besar
        }
    }

    public QuadTreeNode getRoot() {
        return root;
    }

    public List<BufferedImage> getGifFrames() {
        return gifFrames;
    }

    /**
     * Membangun Quadtree; panggil ini di Main.
     */
    public void buildTree() {
        root = buildTreeRecursive(0, 0, image.getWidth(), image.getHeight(),
0);
        // Rekam frame final jika diperlukan
        recordFrame();
    }

    /**
     * Fungsi rekursif untuk membangun Quadtree.
     * Jika error > threshold dan blok masih lebih besar dari minBlockSize,
blok di-split.
     */
    private QuadTreeNode buildTreeRecursive(int x, int y, int width, int
height, int depth) {
        // Pastikan tidak keluar dari batas
        if (x < 0) x = 0;
        if (y < 0) y = 0;
        if (x + width > image.getWidth()) {

```

```

        width = image.getWidth() - x;
    }
    if (y + height > image.getHeight()) {
        height = image.getHeight() - y;
    }
    if (width <= 0 || height <= 0) {
        return null;
    }

    // Hitung rata-rata warna
    Color avgColor = computeAverageColor(x, y, width, height);
    // Hitung error
    double error = errorCalculator.computeError(image, x, y, width,
height, avgColor);

    boolean isBlockTooSmall = (width <= minBlockSize || height <=
minBlockSize);
    boolean errorAcceptable = (error <= threshold);

    // Jika error sudah OK atau blok sudah kecil, jadikan daun
    if (isBlockTooSmall || errorAcceptable) {
        // Rekam frame ketika kita memutuskan daun
        recordFrame();
        return new QuadTreeNode(x, y, width, height, avgColor, error);
    }

    // Kalau masih butuh di-split
    QuadTreeNode node = new QuadTreeNode(x, y, width, height, avgColor,
error);
    node.children = new QuadTreeNode[4];

    // Rekam frame sebelum pembagian
    recordFrame();

    int halfWidth = width / 2;
    int halfHeight = height / 2;
    int width2 = width - halfWidth;      // Menangani ganjil
    int height2 = height - halfHeight; // Menangani ganjil

    // Split 4 sub-blok
    node.children[0] = buildTreeRecursive(x,           y,
halfWidth, halfHeight, depth+1);

```

```

        recordFrame();
        node.children[1] = buildTreeRecursive(x + halfWidth, y,           width2,
halfHeight, depth+1);
        recordFrame();
        node.children[2] = buildTreeRecursive(x,                 y + halfHeight,
halfWidth, height2, depth+1);
        recordFrame();
        node.children[3] = buildTreeRecursive(x + halfWidth, y + halfHeight,
width2, height2, depth+1);
        recordFrame();

        return node;
    }

    /**
     * Merekam satu frame ke gifFrames, menampilkan "batas" di sekitar blok
daun
     * agar terlihat proses splitting.
     * Dengan parameter depth untuk memungkinkan rekaman yang lebih selektif.
     */
private void recordFrame() {
    // Skip jika GIF recording dimatikan
    if (!recordGif) return;

    // Hitung total kemungkinan frame
    totalFramesPossible++;

    // Skip berdasarkan frameSkip
    frameCount++;
    if ((frameCount % frameSkip) != 0) return;

    // Stop jika sudah mencapai batas frame
    if (gifFrames.size() >= maxFrames) return;

    // Untuk mendapatkan distribusi frame yang lebih merata, kita bisa
coba
    // skema sampling berdasarkan prediksi total frame
    if (totalFramesPossible > maxFrames * 4 && gifFrames.size() > 10) {
        // Semakin banyak frame yang dihasilkan, semakin selektif kita
merekam
        if (Math.random() > 0.2) return; // 80% chance to skip additional
frames
}

```

```

    }

    BufferedImage frame = renderTreeState();
    gifFrames.add(frame);
}

/** 
 * Hitung rata-rata warna pada blok.
 */
private Color computeAverageColor(int x, int y, int width, int height) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = 0;

    int endX = x + width;
    int endY = y + height;
    if (endX > image.getWidth()) endX = image.getWidth();
    if (endY > image.getHeight()) endY = image.getHeight();

    for (int j = y; j < endY; j++) {
        for (int i = x; i < endX; i++) {
            Color c = new Color(image.getRGB(i, j));
            sumR += c.getRed();
            sumG += c.getGreen();
            sumB += c.getBlue();
            count++;
        }
    }
    if (count == 0) return Color.BLACK;

    int avgR = (int)(sumR / count);
    int avgG = (int)(sumG / count);
    int avgB = (int)(sumB / count);
    return new Color(avgR, avgG, avgB);
}

/** 
 * Membuat gambar snapshot yang menampilkan boundary node daun
 * di atas background gambar aslinya.
 */
public BufferedImage renderTreeState() {
    // Jika gambarnya sangat besar, buat versi kecilnya untuk GIF
    BufferedImage frame;

```

```

        int maxDimension = 800; // Batasi ukuran frame

        if (image.getWidth() > maxDimension || image.getHeight() >
maxDimension) {
            // Scale down untuk frame GIF
            double scale = (double) maxDimension / Math.max(image.getWidth(),
image.getHeight());
            int newWidth = (int) (image.getWidth() * scale);
            int newHeight = (int) (image.getHeight() * scale);

            frame = new BufferedImage(newWidth, newHeight,
BufferedImage.TYPE_INT_RGB);
            Graphics g = frame.getGraphics();
            g.drawImage(image, 0, 0, newWidth, newHeight, null);

            // Skala juga boundary
            drawScaledBoundaries(g, root, scale);
            g.dispose();
        } else {
            // Untuk gambar yang cukup kecil, gunakan ukuran asli
            frame = new BufferedImage(image.getWidth(), image.getHeight(),
BufferedImage.TYPE_INT_RGB);
            Graphics g = frame.getGraphics();
            g.drawImage(image, 0, 0, null);
            drawBoundaries(g, root);
            g.dispose();
        }

        return frame;
    }

    /**
     * Menggambar kotak merah di setiap node daun dengan skala.
     */
    private void drawScaledBoundaries(Graphics g, QuadTreeNode node, double
scale) {
        if (node == null) return;
        if (node.isLeaf()) {
            g.setColor(Color.RED);
            int x = (int)(node.x * scale);
            int y = (int)(node.y * scale);
            int width = (int)(node.width * scale);

```

```

        int height = (int)(node.height * scale);
        g.drawRect(x, y, width, height);
    } else {
        for (QuadTreeNode child : node.children) {
            drawScaledBoundaries(g, child, scale);
        }
    }
}

/***
 * Menggambar kotak merah di setiap node daun.
 */
private void drawBoundaries(Graphics g, QuadTreeNode node) {
    if (node == null) return;
    if (node.isLeaf()) {
        g.setColor(Color.RED);
        g.drawRect(node.x, node.y, node.width, node.height);
    } else {
        for (QuadTreeNode child : node.children) {
            drawBoundaries(g, child);
        }
    }
}

/***
 * Menghasilkan gambar akhir hasil kompresi,
 * setiap node daun diisi warna rata-ratanya.
 */
public BufferedImage generateCompressedImage() {
    BufferedImage output = new BufferedImage(image.getWidth(),
image.getHeight(), image.getType());
    Graphics g = output.getGraphics();
    fillCompressedImage(g, root);
    g.dispose();
    return output;
}

private void fillCompressedImage(Graphics g, QuadTreeNode node) {
    if (node == null) return;
    if (node.isLeaf()) {
        g.setColor(node.averageColor);
        g.fillRect(node.x, node.y, node.width, node.height);
    }
}

```

```

    } else {
        for (QuadTreeNode child : node.children) {
            fillCompressedImage(g, child);
        }
    }
}

public int countLeaves() {
    return countLeavesRecursive(root);
}

private int countLeavesRecursive(QuadTreeNode node) {
    if (node == null) return 0;
    if (node.isLeaf()) return 1;
    int sum = 0;
    for (QuadTreeNode child : node.children) {
        sum += countLeavesRecursive(child);
    }
    return sum;
}

public int getTreeDepth() {
    return computeDepth(root);
}

private int computeDepth(QuadTreeNode node) {
    if (node == null) return 0;
    if (node.isLeaf()) return 1;
    int max = 0;
    for (QuadTreeNode child : node.children) {
        int d = computeDepth(child);
        if (d > max) max = d;
    }
    return max + 1;
}
}

```

2.10 File QuadTreeNode.java

```
import java.awt.Color;

You, 1 second ago | 2 authors (jovan196 and one other)
public class QuadTreeNode {
    public int x, y, width, height;
    public Color averageColor;
    public double error;
    public QuadTreeNode[] children; // Jika null = daun

    public QuadTreeNode(int x, int y, int width, int height,
                        Color averageColor, double error) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.averageColor = averageColor;
        this.error = error;
        this.children = null;
    }

    public boolean isLeaf() {
        return children == null;
    }
}
```

2.11 File GIFSeqWriter.java

```
● ● ●
```

```
import java.awt.image.RenderedImage;
import java.io.IOException;
import java.util.Iterator;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageTypeSpecifier;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.stream.ImageOutputStream;

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

    public GIFSeqWriter(ImageOutputStream outputStream, int
imageType, int delay, boolean loop) throws 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);

        // Pengaturan "GraphicsControlExtension"
        IIOMetadataNode gce = getNode(root,
"GraphicControlExtension");
        gce.setAttribute("disposalMethod", "none");
        gce.setAttribute("userInputFlag", "FALSE");
        gce.setAttribute("delayTime", Integer.toString(delay /
10));
        gce.setAttribute("transparentColorFlag", "FALSE");

        // Comments
        IIOMetadataNode comments = getNode(root,
"CommentExtensions");
        comments.setAttribute("CommentExtension", "Created by
QuadtreeCompressor");
```

```

        // Loop
        IIOMetadataNode appExtensions = getNode(root,
"ApplicationExtensions");
        IIOMetadataNode child = new
IIOMetadataNode("ApplicationExtension");
        child.setAttribute("applicationID", "NETSCAPE");
        child.setAttribute("authenticationCode", "2.0");
        int loopContinuously = loop ? 0 : 1;
        child.setUserObject(new byte[]{ 0x1, (byte)
(loopContinuously & 0xFF), (byte)((loopContinuously >> 8) &
0xFF)} );
        appExtensions.appendChild(child);

        imageMetaData.setFromTree(metaFormatName, root);

        gifWriter.setOutput(outputStream);
        gifWriter.prepareWriteSequence(null);
    }

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

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

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

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

```

BAGIAN III

Test Case

Test Case	Input	Output
1	 <p>Masukkan lokasi absolut gambar input: Lenna.png Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan pilihan (angka): 1 Masukkan nilai threshold awal: 20 Masukkan ukuran blok minimum: 2 Masukkan target persentase kompresi (floating number, 1.0 = 100%, 0 utk nonaktif): 0</p>	<p>(Zoom jika kurang jelas perbedaannya)</p>  <p>Membuat gambar hasil kompresi... Gambar hasil kompresi berhasil disimpan di: Lenna_out.jpg</p> <pre>===== Hasil Kompresi ===== Waktu eksekusi : 0,378 detik Ukuran gambar sebelum: 473831 bytes Ukuran gambar sesudah: 32276 bytes Persentase kompresi : 95,19% Kedalaman pohon : 9 Banyak simpul (leaf) : 39190 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test></pre>

2



Masukkan lokasi absolut gambar input: gunung.jpg
 Pilih metode perhitungan error:
 1. Variance
 2. Mean Absolute Deviation
 3. Max Pixel Difference
 4. Entropy
 5. Structural Similarity Index (SSIM)
 Masukkan pilihan (angka): 5
 Masukkan nilai threshold awal: 0,7
 Masukkan ukuran blok minimum: 16
 Masukkan target persentase kompresi (floating number, 1.0 = 100%, 0 utk nonaktif): 0



Membuat gambar hasil kompresi...
 Gambar hasil kompresi berhasil disimpan di: gunung_out.jpg
 ===== Hasil Kompresi =====
 Waktu eksekusi : 3,388 detik
 Ukuran gambar sebelum: 989388 bytes
 Ukuran gambar sesudah: 119354 bytes
 Persentase kompresi : 87,94%
 Kedalaman pohon : 8
 Banyak simpul (leaf) : 2839
 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test>

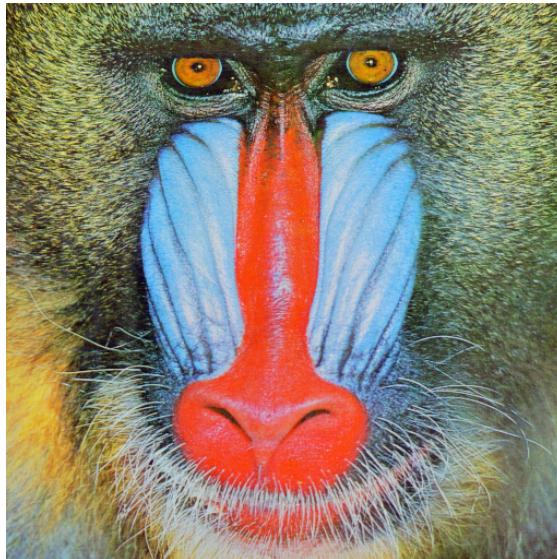


Masukkan lokasi absolut gambar input: bird.bmp
 Pilih metode perhitungan error:
 1. Variance
 2. Mean Absolute Deviation
 3. Max Pixel Difference
 4. Entropy
 5. Structural Similarity Index (SSIM)
 Masukkan pilihan (angka): 2
 Masukkan nilai threshold awal: 10
 Masukkan ukuran blok minimum: 4
 Masukkan target persentase kompresi (floating number, 1.0 = 100%, 0 utk nonaktif): 0,8
 Masukkan lokasi absolut untuk gambar keluaran: bird_out.jpg



Optimal threshold ditemukan: 0.9999990463256836
 Membangun quadtree...
 Membuat gambar hasil kompresi...
 Gambar hasil kompresi berhasil disimpan di: bird_out.jpg
 ===== Hasil Kompresi =====
 Waktu eksekusi : 1,023 detik
 Ukuran gambar sebelum: 66614 bytes
 Ukuran gambar sesudah: 4448 bytes
 Persentase kompresi : 93,32%
 Kedalaman pohon : 7
 Banyak simpul (leaf) : 3067
 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test>

4



Masukkan lokasi absolut gambar input: baboon24.bmp
 Pilih metode perhitungan error:
 1. Variance
 2. Mean Absolute Deviation
 3. Max Pixel Difference
 4. Entropy
 5. Structural Similarity Index (SSIM)
 Masukkan pilihan (angka): 3
 Masukkan nilai threshold awal: 50
 Masukkan ukuran blok minimum: 8
 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0



Membangun quadtree...
 Membuat gambar hasil kompresi...
 Gambar hasil kompresi berhasil disimpan di: baboon_out.jpg

```
===== Hasil Kompresi =====
Waktu eksekusi      : 0,352 detik
Ukuran gambar sebelum: 786486 bytes
Ukuran gambar sesudah: 11050 bytes
Persentase kompresi   : 98,60%
Kedalaman pohon       : 7
Banyak simpul (leaf) : 3982
PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test>
```

5



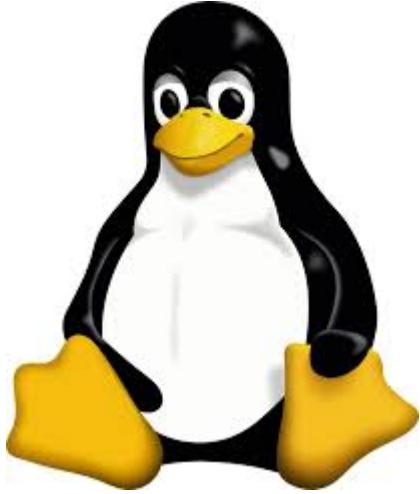
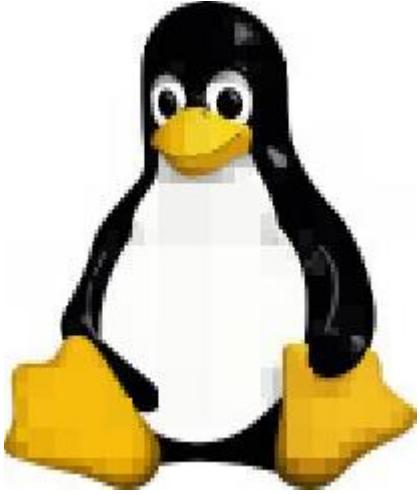
Masukkan lokasi absolut gambar input: boat.bmp
 Pilih metode perhitungan error:
 1. Variance
 2. Mean Absolute Deviation
 3. Max Pixel Difference
 4. Entropy
 5. Structural Similarity Index (SSIM)
 Masukkan pilihan (angka): 4
 Masukkan nilai threshold awal: 2
 Masukkan ukuran blok minimum: 4
 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0
 Masukkan lokasi absolut untuk gambar keluaran: boat_out.jpg



Membangun quadtree...
 Membuat gambar hasil kompresi...
 Gambar hasil kompresi berhasil disimpan di: boat_out.jpg

```
===== Hasil Kompresi =====
Waktu eksekusi      : 0,472 detik
Ukuran gambar sebelum: 263222 bytes
Ukuran gambar sesudah: 20605 bytes
Persentase kompresi   : 92,17%
Kedalaman pohon       : 8
Banyak simpul (leaf) : 16384
PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test>
```

6	 <p>Masukkan lokasi absolut gambar input: peppers512warna.bmp Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan pilihan (angka): 5 Masukkan nilai threshold awal: 0,05 Masukkan ukuran blok minimum: 8 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0</p>	 <p>Membangun quadtree... Membuat gambar hasil kompresi... Gambar hasil kompresi berhasil disimpan di: peppers_out.jpg</p> <p>===== Hasil Kompresi ===== Waktu eksekusi : 0,662 detik Ukuran gambar sebelum: 786486 bytes Ukuran gambar sesudah: 11339 bytes Persentase kompresi : 98,56% Kedalaman pohon : 7 Banyak simpul (leaf) : 4096 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test></p>
7	<p>MT-LEVEL</p> <pre>{spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c fr {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c fr {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c im {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs</pre> <p>Masukkan lokasi absolut gambar input: text.bmp Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan pilihan (angka): 1 Masukkan nilai threshold awal: 500 Masukkan ukuran blok minimum: 2 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0</p>	<p>MT-LEVEL</p> <pre>{spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c fr {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c fr {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs Makefile compress.c fi Makefile~ display.c im {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs {spinet} /home/u/rjkroeger/vfs</pre> <p>Membangun quadtree... Membuat gambar hasil kompresi... Gambar hasil kompresi berhasil disimpan di: text_out.jpg</p> <p>===== Hasil Kompresi ===== Waktu eksekusi : 0,237 detik Ukuran gambar sebelum: 66614 bytes Ukuran gambar sesudah: 19392 bytes Persentase kompresi : 70,89% Kedalaman pohon : 8 Banyak simpul (leaf) : 9553 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test></p>

8	 <pre>Masukkan lokasi absolut gambar input: linux.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan pilihan (angka): 1 Masukkan nilai threshold awal: 500 Masukkan ukuran blok minimum: 2 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0</pre>	 <pre>Membuat gambar hasil kompresi... Gambar hasil kompresi berhasil disimpan di: linux_out.jpg ===== Hasil Kompresi ===== Waktu eksekusi : 0,083 detik Ukuran gambar sebelum: 6497 bytes Ukuran gambar sesudah: 7973 bytes Persentase kompresi : -22,72% Kedalaman pohon : 8 Banyak simpul (leaf) : 2668 Terekam 100 frame untuk animasi GIF. GIF proses kompresi berhasil disimpan di: linux_out.gif PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test></pre>
9	 <pre>Masukkan lokasi absolut gambar input: kincir.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan pilihan (angka): 3 Masukkan nilai threshold awal: 80 Masukkan ukuran blok minimum: 8 Masukkan target persentase kompresi (floating number, 1,0 = 100%, 0 utk nonaktif): 0</pre>	 <pre>Membangun quadtree... Membuat gambar hasil kompresi... Gambar hasil kompresi berhasil disimpan di: kincir_out.jpg ===== Hasil Kompresi ===== Waktu eksekusi : 0,263 detik Ukuran gambar sebelum: 74373 bytes Ukuran gambar sesudah: 41328 bytes Persentase kompresi : 44,43% Kedalaman pohon : 8 Banyak simpul (leaf) : 4957 PS C:\Users\Jovan\Documents\Stima\Tucil2_13523132_13523141\test></pre>

BAGIAN IV

Analisis Algoritma Divide and Conquer

Berikut ini adalah analisis dari algoritma *divide and conquer* yang digunakan dalam program kompresi gambar dengan *quadtree*:

- Pembagian kotak
Sebuah gambar dibagi menjadi empat area atau kotak, sehingga $T(\frac{n}{4})$.
- Rekursi
Tiap kotak atau area (keempat kotak tersebut) dibagi menjadi empat kotak yang lebih kecil lagi secara berulang sampai tidak dapat dibagi lagi, sehingga $4T(\frac{n}{4})$.
- Kalkulasi warna rata-rata
Algoritma ini menghitung warna rata-rata, dengan cara mengambil warna setiap pixel, sehingga merupakan $O(n)$.

Secara keseluruhan, relasi rekurens tersebut adalah

$$T(n) = 4T(\frac{n}{4}) + cn .$$

Berdasarkan Teorema Master, bentuk umum dari relasi rekurens adalah

$$T(n) = aT(\frac{n}{b}) + cn^d .$$

Untuk menentukan kompleksitas algoritma, a dibandingkan dengan b^d . Nilai a adalah 4, sedangkan nilai b^d adalah 4^1 atau 4, sehingga $a = b^d$. Menurut Teorema Master, kompleksitas algoritma dari algoritma divide and conquer yang digunakan pada *quadtree* adalah $O(n \times \log n)$.

BAGIAN V

Bonus

Bagian **target persentase kompresi** (bonus) bertujuan untuk menyesuaikan nilai *threshold* (ambang batas) secara otomatis, menggunakan pencarian biner (binary search) agar rasio kompresi blok (berdasarkan jumlah leaf node dibanding jumlah pixel) mendekati target yang diinginkan (*targetCompression*). *Target compression* di sini diartikan sebagai perbandingan antara jumlah *leaf node* dengan total piksel, di mana semakin sedikit *leaf* yang terbentuk, artinya semakin besar kompresinya.

Langkah-langkah Utama Penyesuaian *Target Compression*:

- Inisialisasi Variabel Pencarian:
 - *Target Compression*: Nilai yang diinginkan (antara 0 dan 1).
 - *Batas Pencarian*: Variabel *low* (0.0) dan *high* (65025 untuk Variance atau 1.0 untuk metode lain).
 - *Threshold Awal*: Dimulai dengan nilai threshold yang sudah ada, disimpan dalam *bestThreshold*.
 - *Toleransi & Iterasi Maksimal*: Toleransi selisih (misalnya 0,01) dan jumlah iterasi maksimal (misalnya 20).

- Loop Binary Search:
 - Hitung nilai tengah (*mid*) dari *low* dan *high* sebagai kandidat threshold.
 - Bangun QuadTree sementara dengan threshold tersebut dan *disable* rekaman GIF untuk efisiensi.
 - Hitung jumlah *leaf node* yang dihasilkan dan tentukan *current compression ratio* dengan rumus:

$$\text{Current Compression Ratio} = 1 - (\text{jumlah leaf} / \text{total pixel})$$

- Bandingkan rasio kompresi saat ini dengan target:
 - Jika selisihnya kurang dari atau sama dengan toleransi, threshold optimal ditemukan.
 - Jika rasio kompresi terlalu rendah (terlalu banyak leaf), artinya gambar belum cukup terkompresi, maka *threshold* perlu diturunkan (set *high* = *mid*).
 - Jika rasio kompresi terlalu tinggi (terlalu sedikit leaf), *threshold* perlu dinaikkan (set *low* = *mid*).
- Update *bestThreshold* dan ulangi hingga iterasi maksimal tercapai atau toleransi terpenuhi.
- Penetapan Threshold Optimal
 - Setelah loop selesai, nilai *bestThreshold* disimpan sebagai threshold final

- Nilai ini akan digunakan dalam pembangunan QuadTree akhir agar struktur hasil kompresi mendekati target persentase yang diharapkan.

SSIM (Structural Similarity Index) merupakan metode pengukuran *error* yang dirancang untuk mengukur kesamaan struktur antara dua gambar, yang sering dipakai untuk mengevaluasi kualitas gambar hasil kompresi atau rekonstruksi. Tidak hanya mengukur perbedaan antarpiksel, SSIM juga memperhitungkan tiga komponen utama:

- Luminance (kecerahan)
- Kontras
- Struktur

Pada implementasi SSIM dalam *SSIMErrorCalc.java*, blok pada gambar asli dibandingkan dengan blok yang direpresentasikan oleh nilai rata-rata (*avgColor*). Karena blok yang direpresentasikan sebagai node pada Quadtree bersifat konstan (hanya satu nilai rata-rata yang digunakan), banyak perhitungan dalam SSIM disederhanakan.

Diberikan dua blok gambar X dan Y berukuran N piksel, SSIM didefinisikan sebagai:

$$SSIM(X, Y) = ((2\mu_X\mu_Y + C_1)(2\sigma_{XY} + C_2)) / ((\mu_X^2 + \mu_Y^2 + C_1)(\sigma_X^2 + \sigma_Y^2 + C_2))$$

Di mana:

μ_X = rata-rata pixel dari blok asli X

μ_Y = rata-rata pixel dari blok rekonstruksi Y

σ_X^2 = varians pixel dari blok asli X

σ_Y^2 = varians pixel dari blok rekonstruksi Y

σ_{XY} = kovarians antara X dan Y

$C_1 = (K_1 L)^2$, $C_2 = (K_2 L)^2$ = konstanta stabilisasi

(biasanya $K_1 = 0.01$; $K_2 = 0.03$; $L = 255$)

BAGIAN VI

Lampiran

Pranala repositori: https://github.com/jovan196/Tucil2_13523132_13523141

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	✓	