LAPORAN TUGAS KECIL STRATEGI ALGORITMA
KOMPRESI GAMBAR MENGGUNAKAN QUADTREE

Laporan untuk memenuhi tugas kecil mata kuliah IF2211 Strategi Algoritma

Disusun Oleh:

M Abizzar Gamadrian – 13523155

INSTITUT TEKNOLOGI BANDUNG

2025

Daftar Isi

Deskripsi Singkat

Laporan ini membahas implementasi algoritma kompresi gambar menggunakan QuadTree, sebuah teknik berbasis Divide and Conquer yang membagi gambar menjadi empat bagian lebih kecil secara rekursif hingga memenuhi kriteria keseragaman tertentu. Algoritma ini dikembangkan menggunakan bahasa pemrograman Java dan memanfaatkan struktur graf untuk merepresentasikan hubungan antara blok-blok hasil dekomposisi gambar.

Proses kompresi dilakukan dengan membangun QuadTree, di mana setiap node merepresentasikan suatu bagian gambar yang dapat dipertahankan atau dibagi lebih lanjut berdasarkan kriteria tertentu. Implementasi ini mengevaluasi kualitas hasil kompresi dengan menggunakan metode pengukuran error seperti MSE (Mean Squared Error) dan PSNR (Peak Signal-to-Noise Ratio) untuk menentukan sejauh mana perbedaan antara gambar asli dan gambar hasil kompresi.

Algoritma Devide and Conquer

Algoritma kompresi gambar dengan metode quadtree menggunakan pendekatan divide and conquer dengan langkah-langkah berikut:

2.1 Divide (Pembagian)

1. Pandang gambar input sebagai satu blok utuh.
2. Hitung nilai error (ketidakseragaman warna) pada blok tersebut menggunakan salah satu metode pengukuran error (Variance, MAD, Max Pixel Difference, atau Entropy).
3. Jika nilai error melebihi threshold yang ditentukan dan ukuran blok masih lebih besar dari minimum block size, bagi blok menjadi empat sub-blok yang sama besar (kuadran).

2.2 Conquer (Penyelesaian)

1. Untuk setiap sub-blok yang dihasilkan, hitung kembali nilai error.
2. Jika error masih di atas threshold dan ukuran blok memungkinkan, lakukan pembagian kembali secara rekursif.
3. Jika error sudah di bawah threshold atau ukuran blok mencapai minimum, simpan blok tersebut sebagai leaf node dengan nilai warna rata-rata dari semua piksel dalam blok tersebut.

2.3 Combine (Penggabungan)

1. Setelah pembagian rekursif selesai, kita memiliki struktur quadtree dengan leaf node yang merepresentasikan blok-blok dengan warna seragam.
2. Rekonstruksi gambar dengan mengganti setiap blok pada leaf node dengan warna rata-ratanya.
3. Hasil akhir adalah gambar yang telah dikompresi di mana area dengan warna seragam direpresentasikan oleh blok besar, sedangkan area dengan detail tinggi direpresentasikan oleh blok-blok kecil.

2.4 Pseudocode Algoritma

Struktur Data QuadTreeNode

```
STRUCT QuadTreeNode:
    x, y: integer          // Koordinat sudut kiri atas
    width, height: integer // Ukuran blok
    avgRed, avgGreen, avgBlue: integer  // Nilai rata-rata warna
    northWest, northEast, southWest, southEast: QuadTreeNode  // Child nodes
    isLeaf: boolean        // Flag apakah node adalah leaf
```

Fungsi BuildQuadTree (Algoritma Divide and Conquer)

```
FUNCTION BuildQuadTree(node, image, threshold, minBlockSize, errorMethod):
    // Hitung rata-rata warna untuk node ini
    CalculateAverageColor(node, image)

    // Hitung error untuk blok ini
    error = ComputeError(image, node.x, node.y, node.width, node.height,
errorMethod)

    // Tentukan apakah blok perlu dibagi (DIVIDE)
    shouldSplit = (error > threshold) AND (node.width > minBlockSize) AND
(node.height > minBlockSize)
                  AND (node.width/2 >= minBlockSize) AND (node.height/2 >=
minBlockSize)

    IF shouldSplit THEN
        // DIVIDE: Bagi node menjadi empat kuadran
        SplitNode(node)

        // CONQUER: Rekursif untuk setiap sub-blok
        BuildQuadTree(node.northWest, image, threshold, minBlockSize,
errorMethod)
        BuildQuadTree(node.northEast, image, threshold, minBlockSize,
errorMethod)
        BuildQuadTree(node.southWest, image, threshold, minBlockSize,
errorMethod)
        BuildQuadTree(node.southEast, image, threshold, minBlockSize,
errorMethod)
    ELSE
        // Simpan sebagai leaf node (blok dengan warna rata-rata)
        node.isLeaf = true
    END IF
END FUNCTION
```

Fungsi SplitNode

```
FUNCTION SplitNode(node):
    newWidth = node.width / 2
    newHeight = node.height / 2

    // Buat empat child nodes untuk kuadran NW, NE, SW, SE
    node.northWest = NEW QuadTreeNode(node.x, node.y, newWidth, newHeight)
    node.northEast = NEW QuadTreeNode(node.x + newWidth, node.y, newWidth,
newHeight)
    node.southWest = NEW QuadTreeNode(node.x, node.y + newHeight, newWidth,
newHeight)
    node.southEast = NEW QuadTreeNode(node.x + newWidth, node.y + newHeight,
newWidth, newHeight)
```

```
    node.isLeaf = false
END FUNCTION
```

Fungsi CalculateAverageColor

```
FUNCTION CalculateAverageColor(node, image):
    sumRed, sumGreen, sumBlue = 0
    pixelCount = 0

    // Iterasi melalui semua piksel dalam blok
    FOR i = node.x TO node.x + node.width - 1 DO
        FOR j = node.y TO node.y + node.height - 1 DO
            IF i < image.width AND j < image.height THEN
                pixel = GetPixel(image, i, j)
                sumRed += GetRed(pixel)
                sumGreen += GetGreen(pixel)
                sumBlue += GetBlue(pixel)
                pixelCount++
            END IF
        END FOR
    END FOR

    // Hitung rata-rata RGB
    IF pixelCount > 0 THEN
        node.avgRed = sumRed / pixelCount
        node.avgGreen = sumGreen / pixelCount
        node.avgBlue = sumBlue / pixelCount
    END IF
END FUNCTION
```

Fungsi ComputeError (untuk Variance)

```
FUNCTION CalculateVariance(image, x, y, width, height):
    // Variance dihitung berdasarkan rumus: σ²rgβ = (σ²r + σ²g + σ²β)/3
    // di mana σ²r = (1/N)∑(Pi,r - μr)²

    // 1. Hitung rata-rata untuk setiap kanal warna
    avgValues = CalculateAverageRGB(image, x, y, width, height)
    avgRed = avgValues[0]
    avgGreen = avgValues[1]
    avgBlue = avgValues[2]

    // 2. Inisialisasi variabel untuk menghitung variansi
    sumRedVariance = 0
    sumGreenVariance = 0
    sumBlueVariance = 0
    pixelCount = 0

    // 3. Hitung sum of squared differences untuk setiap piksel
    FOR i = x TO x + width - 1 DO
        FOR j = y TO y + height - 1 DO
            IF i < image.width AND j < image.height THEN
                pixel = GetPixel(image, i, j)

                // Hitung selisih kuadrat untuk masing-masing kanal
                redDiff = GetRed(pixel) - avgRed
                greenDiff = GetGreen(pixel) - avgGreen
                blueDiff = GetBlue(pixel) - avgBlue

                sumRedVariance += redDiff * redDiff    // (Pi,r - μr)²
                sumGreenVariance += greenDiff * greenDiff  // (Pi,g - μg)²
                sumBlueVariance += blueDiff * blueDiff   // (Pi,β - μβ)²

                pixelCount++
            END IF
```

```
        END FOR
    END FOR

    // 4. Hitung variansi untuk setiap kanal warna
    varianceRed = 0
    varianceGreen = 0
    varianceBlue = 0

    IF pixelCount > 0 THEN
        varianceRed = sumRedVariance / pixelCount      // σ²r = (1/N)∑(Pi,r -
μr)²
        varianceGreen = sumGreenVariance / pixelCount  // σ²g = (1/N)∑(Pi,g -
μg)²
        varianceBlue = sumBlueVariance / pixelCount    // σ²β = (1/N)∑(Pi,β -
μβ)²
    END IF

    // 5. Hitung rata-rata variansi dari ketiga kanal
    averageVariance = (varianceRed + varianceGreen + varianceBlue) / 3.0  //
σ²rgβ = (σ²r + σ²g + σ²β)/3

    RETURN averageVariance
END FUNCTION

FUNCTION CalculateMAD(image, x, y, width, height):
    // MAD dihitung berdasarkan rumus: MADrgβ = (MADr + MADg + MADβ)/3
    // di mana MADr = (1/N)∑|Pi,r - μr|

    // 1. Hitung rata-rata untuk setiap kanal warna
    avgValues = CalculateAverageRGB(image, x, y, width, height)
    avgRed = avgValues[0]
    avgGreen = avgValues[1]
    avgBlue = avgValues[2]

    // 2. Inisialisasi variabel untuk menghitung MAD
    sumRedMAD = 0
    sumGreenMAD = 0
    sumBlueMAD = 0
    pixelCount = 0

    // 3. Hitung sum of absolute differences untuk setiap piksel
    FOR i = x TO x + width - 1 DO
        FOR j = y TO y + height - 1 DO
            IF i < image.width AND j < image.height THEN
                pixel = GetPixel(image, i, j)

                // Hitung selisih absolut untuk masing-masing kanal
                redDiff = ABS(GetRed(pixel) - avgRed)      // |Pi,r - μr|
                greenDiff = ABS(GetGreen(pixel) - avgGreen)  // |Pi,g - μg|
                blueDiff = ABS(GetBlue(pixel) - avgBlue)    // |Pi,β - μβ|

                sumRedMAD += redDiff
                sumGreenMAD += greenDiff
                sumBlueMAD += blueDiff

                pixelCount++
            END IF
        END FOR
    END FOR

    // 4. Hitung MAD untuk setiap kanal warna
    madRed = 0
    madGreen = 0
    madBlue = 0

    IF pixelCount > 0 THEN
        madRed = sumRedMAD / pixelCount      // MADr = (1/N)∑|Pi,r - μr|
        madGreen = sumGreenMAD / pixelCount  // MADg = (1/N)∑|Pi,g - μg|
```

```
        madBlue = sumBlueMAD / pixelCount    // MADβ = (1/N)∑|Pi,β - μβ|
    END IF

    // 5. Hitung rata-rata MAD dari ketiga kanal
    averageMAD = (madRed + madGreen + madBlue) / 3.0  // MADrgβ = (MADr + MADg +
MADβ)/3

    RETURN averageMAD
END FUNCTION

FUNCTION CalculateMaxPixelDifference(image, x, y, width, height):
    // Max Pixel Difference dihitung berdasarkan rumus: Drgβ = (Dr + Dg + Dβ)/3
    // di mana Dr = max(Pi,r) - min(Pi,r)

    // 1. Inisialisasi variabel untuk mencari nilai minimum dan maksimum
    minRed = 255    // Nilai maksimum untuk channel 8-bit
    minGreen = 255
    minBlue = 255

    maxRed = 0      // Nilai minimum untuk channel 8-bit
    maxGreen = 0
    maxBlue = 0

    // 2. Cari nilai minimum dan maksimum untuk setiap kanal warna
    FOR i = x TO x + width - 1 DO
        FOR j = y TO y + height - 1 DO
            IF i < image.width AND j < image.height THEN
                pixel = GetPixel(image, i, j)

                red = GetRed(pixel)
                green = GetGreen(pixel)
                blue = GetBlue(pixel)

                // Update nilai minimum
                minRed = MIN(minRed, red)
                minGreen = MIN(minGreen, green)
                minBlue = MIN(minBlue, blue)

                // Update nilai maksimum
                maxRed = MAX(maxRed, red)
                maxGreen = MAX(maxGreen, green)
                maxBlue = MAX(maxBlue, blue)
            END IF
        END FOR
    END FOR

    // 3. Hitung selisih antara nilai maksimum dan minimum untuk setiap kanal
    diffRed = maxRed - minRed        // Dr = max(Pi,r) - min(Pi,r)
    diffGreen = maxGreen - minGreen  // Dg = max(Pi,g) - min(Pi,g)
    diffBlue = maxBlue - minBlue     // Dβ = max(Pi,β) - min(Pi,β)

    // 4. Hitung rata-rata selisih dari ketiga kanal
    averageDiff = (diffRed + diffGreen + diffBlue) / 3.0  // Drgβ = (Dr + Dg +
Dβ)/3

    RETURN averageDiff
END FUNCTION

FUNCTION CalculateEntropy(image, x, y, width, height):
    // Entropy dihitung berdasarkan rumus: Hrgβ = (Hr + Hg + Hβ)/3
    // di mana Hr = -∑Pr(i)log2(Pr(i))

    // 1. Inisialisasi histogram untuk masing-masing kanal (nilai 0-255)
    histRed = NEW ARRAY[256] OF Integer    // Inisialisasi semua elemen dengan 0
    histGreen = NEW ARRAY[256] OF Integer
    histBlue = NEW ARRAY[256] OF Integer

    pixelCount = 0
```

```
    // 2. Bangun histogram untuk setiap kanal warna
    FOR i = x TO x + width - 1 DO
        FOR j = y TO y + height - 1 DO
            IF i < image.width AND j < image.height THEN
                pixel = GetPixel(image, i, j)

                // Inkrement histogram berdasarkan nilai masing-masing kanal
                histRed[GetRed(pixel)]++
                histGreen[GetGreen(pixel)]++
                histBlue[GetBlue(pixel)]++

                pixelCount++
            END IF
        END FOR
    END FOR

    // 3. Hitung entropy untuk setiap kanal warna
    entropyRed = 0.0
    entropyGreen = 0.0
    entropyBlue = 0.0

    IF pixelCount > 0 THEN
        // Hitung entropy untuk kanal red
        FOR i = 0 TO 255 DO
            IF histRed[i] > 0 THEN
                // Hitung probabilitas pixel dengan nilai i
                probability = histRed[i] / pixelCount  // Pr(i)

                // Akumulasi entropy menggunakan formula Shannon
                entropyRed -= probability * LOG2(probability)  // -
Pr(i)log2(Pr(i))
            END IF
        END FOR

        // Hitung entropy untuk kanal green
        FOR i = 0 TO 255 DO
            IF histGreen[i] > 0 THEN
                probability = histGreen[i] / pixelCount  // Pg(i)
                entropyGreen -= probability * LOG2(probability)  // -
Pg(i)log2(Pg(i))
            END IF
        END FOR

        // Hitung entropy untuk kanal blue
        FOR i = 0 TO 255 DO
            IF histBlue[i] > 0 THEN
                probability = histBlue[i] / pixelCount  // Pβ(i)
                entropyBlue -= probability * LOG2(probability)  // -
Pβ(i)log2(Pβ(i))
            END IF
        END FOR
    END IF

    // 4. Hitung rata-rata entropy dari ketiga kanal
    averageEntropy = (entropyRed + entropyGreen + entropyBlue) / 3.0  // Hrgβ =
(Hr + Hg + Hβ)/3

    RETURN averageEntropy
END FUNCTION

// Helper function untuk menghitung logaritma basis 2
FUNCTION LOG2(value):
    // Log2(x) = ln(x) / ln(2)
    RETURN LN(value) / LN(2)
END FUNCTION
```

Fungsi CompressImage

```
FUNCTION CompressImage(inputImage, threshold, minBlockSize, errorMethod):
    // Inisialisasi
    rootNode = NEW QuadTreeNode(0, 0, inputImage.width, inputImage.height)

    // Bangun quadtree
    BuildQuadTree(rootNode, inputImage, threshold, minBlockSize, errorMethod)

    // Rekonstruksi gambar terkompresi
    compressedImage = ReconstructImage(rootNode, inputImage.width,
inputImage.height)

    RETURN compressedImage, rootNode
END FUNCTION
```

Fungsi ReconstructImage

```
FUNCTION ReconstructImage(rootNode, width, height):
    // Buat gambar kosong dengan ukuran yang sama
    compressedImage = NEW Image(width, height)

    // Rekursif mengisi gambar berdasarkan quadtree
    FillImageFromNode(rootNode, compressedImage)

    RETURN compressedImage
END FUNCTION

FUNCTION FillImageFromNode(node, image):
    IF node.isLeaf THEN
        // Untuk leaf node, isi blok dengan warna rata-rata
        avgColor = CreateColor(node.avgRed, node.avgGreen, node.avgBlue)

        FOR i = node.x TO node.x + node.width - 1 DO
            FOR j = node.y TO node.y + node.height - 1 DO
                IF i < image.width AND j < image.height THEN
                    SetPixel(image, i, j, avgColor)
                END IF
            END FOR
        END FOR
    ELSE
        // Untuk non-leaf node, rekursif ke child nodes
        FillImageFromNode(node.northWest, image)
        FillImageFromNode(node.northEast, image)
        FillImageFromNode(node.southWest, image)
        FillImageFromNode(node.southEast, image)
    END IF
END FUNCTION
```

Struktur Program

Program kompresi gambar quadtree diimplementasikan dalam bahasa Java dengan struktur sebagai berikut:

Kelas-kelas Utama

QuadTreeNode.java - Representasi node dalam struktur quadtree

- Menyimpan informasi posisi (x, y), ukuran (width, height)
- Menyimpan nilai rata-rata warna RGB

- Menyediakan metode untuk split (membagi node) dan calculate average (menghitung rata-rata warna)

ErrorMeasurement.java - Interface untuk metode pengukuran error

- Mendefinisikan kontrak untuk menghitung error pada blok gambar

ErrorMeasurementImpl.java - Implementasi metode pengukuran error

- Mengimplementasikan berbagai metode perhitungan error (Variance, MAD, Max Pixel Difference, Entropy)

ImageProcessor.java - Utilitas pemrosesan gambar

- Menyediakan metode untuk normalisasi warna blok
- Menghitung statistik quadtree (kedalaman, jumlah node

QuadTreeCompression.java - Kelas utama program

- Mengatur alur kompresi gambar
- Membangun quadtree menggunakan algoritma divide and conquer
- Menyimpan gambar hasil kompresi dan menampilkan statistik

Alur Program

1) Baca gambar input dari file
2) Inisialisasi parameter (metode error, threshold, minimum block size)
3) Bangun quadtree dengan algoritma divide and conquer
4) Rekonstruksi gambar berdasarkan quadtree yang dihasilkan
5) Simpan gambar hasil kompresi
6) Hitung dan tampilkan statistik (waktu eksekusi, kedalaman tree, jumlah node, persentase kompresi)

Source code

QuadTreeNode.java

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


public class QuadTreeNode {
    // Posisi dan ukuran blok
    private final int x;        // Koordinat sudut kiri atas
    private final int y;        // Koordinat sudut kiri atas
    private final int width; // Lebar dan tinggi blok
    private final int height; // Lebar dan tinggi blok

    // Nilai rata-rata RGB untuk blok ini
    private int avgRed;
    private int avgGreen;
    private int avgBlue;
```

```java
    // Child nodes (NW, NE, SW, SE)
    private QuadTreeNode northWest;
    private QuadTreeNode northEast;
    private QuadTreeNode southWest;
    private QuadTreeNode southEast;

    // Flag yang menunjukkan apakah node ini adalah leaf (blok yang tidak dibagi lagi)
    private boolean isLeaf;

    /**
     * Constructor untuk membuat node baru
     */
    public QuadTreeNode(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.isLeaf = true; // Default sebagai leaf node
    }

    /**
     * Menghitung nilai rata-rata RGB dari semua piksel dalam blok
     */
    // TODO: Cek lagi ini, kadang masih ada bug kalo imagenya ukuran aneh
    public void calculateAverage(BufferedImage image) {
        long sumRed = 0, sumGreen = 0, sumBlue = 0;
        int pixelCount = 0;

        // Iterasi melalui semua piksel dalam blok ini
        for (int i = x; i < x + width && i < image.getWidth(); i++) {
            for (int j = y; j < y + height && j < image.getHeight(); j++) {
                Color color = new Color(image.getRGB(i, j), true); // true untuk mempertahankan alpha
                sumRed += color.getRed();
                sumGreen += color.getGreen();
                sumBlue += color.getBlue();
                pixelCount++;
            }
        }

        // Hitung rata-rata dengan pembulatan yang lebih akurat
        if (pixelCount > 0) {
            // Gunakan Math.round dan casting ke double untuk menghindari masalah pembulatan
            avgRed = (int) Math.round((double)sumRed / pixelCount);
            avgGreen = (int) Math.round((double)sumGreen / pixelCount);
            avgBlue = (int) Math.round((double)sumBlue / pixelCount);

            // Pastikan nilai berada dalam rentang valid 0-255
            avgRed = Math.max(0, Math.min(255, avgRed));
```

```
            avgGreen = Math.max(0, Math.min(255, avgGreen));
            avgBlue = Math.max(0, Math.min(255, avgBlue));
        }
    }

    /**
     * Membagi node menjadi empat anak (children)
     */
    public void split() {
        // Hitung ukuran baru dengan mempertimbangkan piksel terakhir
        int newWidth = (width + 1) / 2;
        int newHeight = (height + 1) / 2;

        // Width dan height kuadran timur/selatan mungkin berbeda
        int eastWidth = width - newWidth;
        int southHeight = height - newHeight;

        // Buat empat quadran dengan ukuran yang benar
        northWest = new QuadTreeNode(x, y, newWidth, newHeight);
        northEast = new QuadTreeNode(x + newWidth, y, eastWidth, newHeight);
        southWest = new QuadTreeNode(x, y + newHeight, newWidth, southHeight);
        southEast = new QuadTreeNode(x + newWidth, y + newHeight, eastWidth, southHeight);

        isLeaf = false;
    }

    // Getter dan setter untuk properti
    public boolean isLeaf() {
        return isLeaf;
    }

    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public Color getAverageColor() {
```

```
        return new Color(avgRed, avgGreen, avgBlue);
    }

    public QuadTreeNode getNorthWest() {
        return northWest;
    }

    public QuadTreeNode getNorthEast() {
        return northEast;
    }

    public QuadTreeNode getSouthWest() {
        return southWest;
    }

    public QuadTreeNode getSouthEast() {
        return southEast;
    }
}
```

ErrorMeasurementImpl.java

```
import java.awt.Color;
import java.awt.image.BufferedImage;
/**
 * Implementasi dari metode pengukuran error
 */
public class ErrorMeasurementImpl implements ErrorMeasurement {
    public static final int VARIANCE = 1;
    public static final int MEAN_ABSOLUTE_DEVIATION = 2;
    public static final int MAX_PIXEL_DIFFERENCE = 3;
    public static final int ENTROPY = 4;

    private final int method;

    public ErrorMeasurementImpl(int method) {
        this.method = method;
    }

    @Override
    public double calculateError(BufferedImage image, int x, int y, int width, int height) {
        return switch (method) {
            case VARIANCE -> calculateVariance(image, x, y, width, height);
            case MEAN_ABSOLUTE_DEVIATION -> calculateMAD(image, x, y, width, height);
            case MAX_PIXEL_DIFFERENCE -> calculateMaxPixelDifference(image, x, y, width, height);
            case ENTROPY -> calculateEntropy(image, x, y, width, height);
            default -> calculateVariance(image, x, y, width, height);
        };
```

```java
}

/**
 * Menghitung variansi sesuai rumus dalam tugas
 */
private double calculateVariance(BufferedImage image, int x, int y, int width, int height) {
    // Hitung rata-rata untuk setiap kanal warna
    double[] avgValues = calculateAverages(image, x, y, width, height);
    double avgRed = avgValues[0];
    double avgGreen = avgValues[1];
    double avgBlue = avgValues[2];

    double sumRedVariance = 0;
    double sumGreenVariance = 0;
    double sumBlueVariance = 0;
    int pixelCount = 0;

    // Hitung sum of squared differences
    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            Color color = new Color(image.getRGB(i, j));

            sumRedVariance += Math.pow(color.getRed() - avgRed, 2);
            sumGreenVariance += Math.pow(color.getGreen() - avgGreen, 2);
            sumBlueVariance += Math.pow(color.getBlue() - avgBlue, 2);

            pixelCount++;
        }
    }

    // Hitung variansi setiap kanal
    double varianceRed = pixelCount > 0 ? sumRedVariance / pixelCount : 0;
    double varianceGreen = pixelCount > 0 ? sumGreenVariance / pixelCount : 0;
    double varianceBlue = pixelCount > 0 ? sumBlueVariance / pixelCount : 0;

    // Rata-rata variansi dari ketiga kanal
    return (varianceRed + varianceGreen + varianceBlue) / 3.0;
}

/**
 * Menghitung Mean Absolute Deviation sesuai rumus
 */
private double calculateMAD(BufferedImage image, int x, int y, int width, int height) {
    // Hitung rata-rata untuk setiap kanal warna
    double[] avgValues = calculateAverages(image, x, y, width, height);
    double avgRed = avgValues[0];
    double avgGreen = avgValues[1];
    double avgBlue = avgValues[2];
```

```
    double sumRedMAD = 0;
    double sumGreenMAD = 0;
    double sumBlueMAD = 0;
    int pixelCount = 0;

    // Hitung sum of absolute differences
    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            Color color = new Color(image.getRGB(i, j));

            sumRedMAD += Math.abs(color.getRed() - avgRed);
            sumGreenMAD += Math.abs(color.getGreen() - avgGreen);
            sumBlueMAD += Math.abs(color.getBlue() - avgBlue);

            pixelCount++;
        }
    }

    // Hitung MAD setiap kanal
    double madRed = pixelCount > 0 ? sumRedMAD / pixelCount : 0;
    double madGreen = pixelCount > 0 ? sumGreenMAD / pixelCount : 0;
    double madBlue = pixelCount > 0 ? sumBlueMAD / pixelCount : 0;

    // Rata-rata MAD dari ketiga kanal
    return (madRed + madGreen + madBlue) / 3.0;
}

/**
 * Menghitung Max Pixel Difference sesuai rumus
 */
private double calculateMaxPixelDifference(BufferedImage image, int x, int y, int width, int height) {
    int minRed = 255, minGreen = 255, minBlue = 255;
    int maxRed = 0, maxGreen = 0, maxBlue = 0;

    // Cari nilai min dan max untuk setiap kanal
    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            Color color = new Color(image.getRGB(i, j));

            minRed = Math.min(minRed, color.getRed());
            minGreen = Math.min(minGreen, color.getGreen());
            minBlue = Math.min(minBlue, color.getBlue());

            maxRed = Math.max(maxRed, color.getRed());
            maxGreen = Math.max(maxGreen, color.getGreen());
            maxBlue = Math.max(maxBlue, color.getBlue());
        }
```

```
    }

    // Hitung selisih max-min setiap kanal
    double diffRed = maxRed - minRed;
    double diffGreen = maxGreen - minGreen;
    double diffBlue = maxBlue - minBlue;

    // Rata-rata selisih dari ketiga kanal
    return (diffRed + diffGreen + diffBlue) / 3.0;
}

/**
 * Menghitung Entropy sesuai rumus
 */
private double calculateEntropy(BufferedImage image, int x, int y, int width, int height) {
    // Hitung histogram untuk setiap kanal (0-255)
    int[] histRed = new int[256];
    int[] histGreen = new int[256];
    int[] histBlue = new int[256];
    int pixelCount = 0;

    // Bangun histogram
    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            Color color = new Color(image.getRGB(i, j));

            histRed[color.getRed()]++;
            histGreen[color.getGreen()]++;
            histBlue[color.getBlue()]++;

            pixelCount++;
        }
    }

    // Hitung entropy untuk setiap kanal
    double entropyRed = 0;
    double entropyGreen = 0;
    double entropyBlue = 0;

    if (pixelCount > 0) {
        for (int i = 0; i < 256; i++) {
            if (histRed[i] > 0) {
                double probabilityRed = (double) histRed[i] / pixelCount;
                entropyRed -= probabilityRed * (Math.log(probabilityRed) / Math.log(2));
            }

            if (histGreen[i] > 0) {
                double probabilityGreen = (double) histGreen[i] / pixelCount;
```

```
                    entropyGreen -= probabilityGreen * (Math.log(probabilityGreen) / Math.log(2));
                }

                if (histBlue[i] > 0) {
                    double probabilityBlue = (double) histBlue[i] / pixelCount;
                    entropyBlue -= probabilityBlue * (Math.log(probabilityBlue) / Math.log(2));
                }
            }
        }

        // Rata-rata entropy dari ketiga kanal
        return (entropyRed + entropyGreen + entropyBlue) / 3.0;
    }

    /**
     * Helper method untuk menghitung rata-rata RGB
     */
    private double[] calculateAverages(BufferedImage image, int x, int y, int width, int height) {
        long sumRed = 0, sumGreen = 0, sumBlue = 0;
        int pixelCount = 0;

        for (int i = x; i < x + width && i < image.getWidth(); i++) {
            for (int j = y; j < y + height && j < image.getHeight(); j++) {
                Color color = new Color(image.getRGB(i, j));
                sumRed += color.getRed();
                sumGreen += color.getGreen();
                sumBlue += color.getBlue();
                pixelCount++;
            }
        }

        double[] averages = new double[3];
        if (pixelCount > 0) {
            averages[0] = (double) sumRed / pixelCount;
            averages[1] = (double) sumGreen / pixelCount;
            averages[2] = (double) sumBlue / pixelCount;
        }

        return averages;
    }
}
```

ErrorMeasurement.java

```
import java.awt.image.BufferedImage;
/**
 * Interface untuk metode pengukuran error
 */
```

```java
public interface ErrorMeasurement {
    /**
     * Menghitung error untuk blok gambar tertentu
     *
     * @param image Gambar yang diproses
     * @param x Koordinat X awal blok
     * @param y Koordinat Y awal blok
     * @param width Lebar blok
     * @param height Tinggi blok
     * @return Nilai error (semakin tinggi berarti semakin tidak seragam)
     */
    double calculateError(BufferedImage image, int x, int y, int width, int height);
}
```

ImageProcessor.java

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

/**
 * Kelas untuk memproses dan memanipulasi gambar
 */
public class ImageProcessor {
    /**
     * Menormalisasi warna dalam blok dengan menggunakan rata-rata RGB
     */
    public static void normalizeBlock(BufferedImage image, QuadTreeNode node) {
        if (node.isLeaf()) {
            // Hanya normalisasi leaf nodes
            Color avgColor = node.getAverageColor();
            int rgbValue = avgColor.getRGB();

            // Set semua piksel dalam blok ke warna rata-rata
            for (int i = node.getX(); i < node.getX() + node.getWidth() && i < image.getWidth(); i++) {
                for (int j = node.getY(); j < node.getY() + node.getHeight() && j < image.getHeight(); j++) {
                    image.setRGB(i, j, rgbValue);
                }
            }
        } else {
            // Rekursi untuk semua child nodes
            normalizeBlock(image, node.getNorthWest());
            normalizeBlock(image, node.getNorthEast());
            normalizeBlock(image, node.getSouthWest());
            normalizeBlock(image, node.getSouthEast());
        }
    }

    /**
```

```
    * Menghitung kedalaman pohon quadtree
    */
    public static int calculateTreeDepth(QuadTreeNode node) {
        if (node == null) {
            return 0;
        }

        if (node.isLeaf()) {
            return 1;
        }

        // Hitung kedalaman maksimum dari semua anak
        int nwDepth = calculateTreeDepth(node.getNorthWest());
        int neDepth = calculateTreeDepth(node.getNorthEast());
        int swDepth = calculateTreeDepth(node.getSouthWest());
        int seDepth = calculateTreeDepth(node.getSouthEast());

        return 1 + Math.max(Math.max(nwDepth, neDepth), Math.max(swDepth, seDepth));
    }

    /**
     * Menghitung jumlah node dalam pohon quadtree
     */
    public static int countNodes(QuadTreeNode node) {
        if (node == null) {
            return 0;
        }

        if (node.isLeaf()) {
            return 1;
        }

        // Hitung jumlah node untuk semua anak dan tambahkan node ini
        return 1 + countNodes(node.getNorthWest()) +
                   countNodes(node.getNorthEast()) +
                   countNodes(node.getSouthWest()) +
                   countNodes(node.getSouthEast());
    }
}
```

QuadTreeCompression.java (main program)

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

```java
 * Kelas utama untuk proses kompresi gambar dengan quadtree
 */
public class QuadTreeCompression {
    private BufferedImage originalImage;
    private BufferedImage compressedImage;
    private QuadTreeNode rootNode;
    private ErrorMeasurement errorMeasurement;
    private String originalImagePath;

    private int minBlockSize;
    private double threshold;
    private int treeDepth;
    private int nodeCount;
    private long executionTime;

    // Konstruktor
    public QuadTreeCompression(String inputImagePath, int errorMethod, double threshold, int minBlockSize) {
        try {
            this.originalImagePath = inputImagePath;
            this.originalImage = ImageIO.read(new File(inputImagePath));
            this.compressedImage = new BufferedImage(
                originalImage.getWidth(),
                originalImage.getHeight(),
                originalImage.getType()
            );

            // Salin gambar asli ke gambar terkompresi sebagai permulaan
            for (int x = 0; x < originalImage.getWidth(); x++) {
                for (int y = 0; y < originalImage.getHeight(); y++) {
                    compressedImage.setRGB(x, y, originalImage.getRGB(x, y));
                }
            }

            this.errorMeasurement = new ErrorMeasurementImpl(errorMethod);
            this.threshold = threshold;
            this.minBlockSize = minBlockSize;
        } catch (IOException e) {
            System.err.println("Error loading image: " + e.getMessage());
        }
    }

    /**
     * Memulai proses kompresi
     */
    public void compress() {
        long startTime = System.currentTimeMillis();

        // Buat node root untuk seluruh gambar
```

```
        rootNode = new QuadTreeNode(0, 0, originalImage.getWidth(), originalImage.getHeight());

        // Bangun quadtree dengan algoritma divide and conquer
        buildQuadTree(rootNode, originalImage);

        // Normalisasi warna untuk setiap blok
        ImageProcessor.normalizeBlock(compressedImage, rootNode);

        // Hitung statistik
        treeDepth = ImageProcessor.calculateTreeDepth(rootNode);
        nodeCount = ImageProcessor.countNodes(rootNode);

        executionTime = System.currentTimeMillis() - startTime;
    }

    /**
     * Algoritma rekursif divide and conquer untuk membangun quadtree
     */
    private void buildQuadTree(QuadTreeNode node, BufferedImage image) {
        // Hitung nilai rata-rata RGB untuk node ini
        node.calculateAverage(image);

        // Hitung error untuk node ini
        double error = errorMeasurement.calculateError(
            image, node.getX(), node.getY(), node.getWidth(), node.getHeight()
        );

        // Kondisi untuk membagi atau tidak:
        // 1. Error di atas threshold
        // 2. Ukuran blok lebih besar dari minimum block size
        // 3. Ukuran blok setelah dibagi tidak kurang dari minimum block size
        boolean shouldSplit = error > threshold &&
                              node.getWidth() > minBlockSize &&
                              node.getHeight() > minBlockSize &&
                              node.getWidth()/2 >= minBlockSize &&
                              node.getHeight()/2 >= minBlockSize;

        if (shouldSplit) {
            // Bagi node menjadi empat
            node.split();

            // Rekursif untuk setiap anak node
            buildQuadTree(node.getNorthWest(), image);
            buildQuadTree(node.getNorthEast(), image);
            buildQuadTree(node.getSouthWest(), image);
            buildQuadTree(node.getSouthEast(), image);
        }
        // Jika tidak dibagi, node ini menjadi leaf node dengan warna rata-rata
```

```
    }

    /**
     * Menyimpan gambar hasil kompresi
     */
    public void saveCompressedImage(String outputPath) {
        try {
            String extension = outputPath.substring(outputPath.lastIndexOf('.') + 1);
            File outputFile = new File(outputPath);
            ImageIO.write(compressedImage, extension, outputFile);
            System.out.println("Compressed image saved to: " + outputPath);
        } catch (IOException e) {
            System.err.println("Error saving compressed image: " + e.getMessage());
        }
    }

    /**
     * Menghitung dan mencetak statistik kompresi
     */
    public void printStatistics() {
        System.out.println("Waktu Eksekusi: " + executionTime + " ms");
        System.out.println("Kedalaman Pohon: " + treeDepth);
        System.out.println("Jumlah Nodes: " + nodeCount);

        // Hitung dan tampilkan persentase kompresi
        File originalFile = new File(originalImagePath);
        long originalSize = originalFile.length();
        long compressedSize = 0; // Ini harus dihitung setelah menyimpan file

        File tempOutput = new File("temp_compressed.png");
        try {
            ImageIO.write(compressedImage, "png", tempOutput);
            compressedSize = tempOutput.length();
            tempOutput.delete();
        } catch (IOException e) {
        }

        double compressionPercentage = (1.0 - (double)compressedSize / originalSize) * 100;

        System.out.println("Ukuran Gambar Asli: " + originalSize + " bytes");
        System.out.println("Ukuran Gambar Terkompresi: " + compressedSize + " bytes");
        System.out.println("Persentase Kompresi: " + String.format("%.2f", compressionPercentage) + "%");
    }


/**
 * Method utama untuk menjalankan program
 */
```

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

    System.out.println("Image Compression using Quadtree");
    System.out.println("================================");

    // Input dari pengguna
    String inputPath = "";
    while (inputPath.isEmpty()) {
        try {
            System.out.print("Masukkan absolute path untuk input gambar: ");
            inputPath = scanner.nextLine().trim();
            File inputFile = new File(inputPath);
            if (!inputFile.exists() || !inputFile.isFile()) {
                System.out.println("Error: File yang dimasksud tidak ada, tolong coba lagi.");
                inputPath = "";
            }
        } catch (Exception e) {
            System.out.println("Error saat membaca input path. Tolong coba lagi.");
            inputPath = "";
        }
    }

    // Input metode error
    int errorMethod = 0;
    while (errorMethod < 1 || errorMethod > 4) {
        try {
            System.out.println("Pilih error measurement method:");
            System.out.println("1. Variance");
            System.out.println("2. Mean Absolute Deviation (MAD)");
            System.out.println("3. Max Pixel Difference");
            System.out.println("4. Entropy");
            System.out.print("Masukkan Pilihanmu (1-4): ");

            String input = scanner.nextLine().trim();
            errorMethod = Integer.parseInt(input);

            if (errorMethod < 1 || errorMethod > 4) {
                System.out.println("Error: Angka yang dimasukkan harus bernilai antara 1 sampai 4");
            }
        } catch (NumberFormatException e) {
            System.out.println("Error: Tolong masukkan angka yang valid.");
        } catch (Exception e) {
            System.out.println("An unexpected error occurred: " + e.getMessage());
        }
    }

    // Input threshold
```

```
double threshold = -1;
while (threshold < 0) {
    try {
        System.out.print("Masukkan nilai threshold: ");
        String thresholdStr = scanner.nextLine().trim();
        // Ganti koma dengan titik jika diperlukan
        thresholdStr = thresholdStr.replace(',', '.');
        threshold = Double.parseDouble(thresholdStr);

        if (threshold < 0) {
            System.out.println("Error: Nilai Threshold haruslah positif.");
        }
    } catch (NumberFormatException e) {
        System.out.println("Error: Masukkan angka yang valid untuk Threshold.");
    } catch (Exception e) {
        System.out.println("An unexpected error occurred: " + e.getMessage());
    }
}

// Input minimum block size
int minBlockSize = 0;
while (minBlockSize <= 0) {
    try {
        System.out.print("Masukkan minimum block size: ");
        String minBlockSizeStr = scanner.nextLine().trim();
        minBlockSize = Integer.parseInt(minBlockSizeStr);

        if (minBlockSize <= 0) {
            System.out.println("Error: Minimum block size harus berupa bilangan bulat positif!");
        }
    } catch (NumberFormatException e) {
        System.out.println("Error: Tolong masukkan bilangan bulat yang valid untuk minimum block size.");
    } catch (Exception e) {
        System.out.println("An unexpected error occurred: " + e.getMessage());
    }
}

// Input output path
String outputPath = "";
while (outputPath.isEmpty()) {
    try {
        System.out.print("Masukkan absolute path untuk output gambar terkompresi: ");
        outputPath = scanner.nextLine().trim();

        // Validasi bahwa outputPath memiliki ekstensi valid
        if (!outputPath.matches(".*\\.(jpg|jpeg|png|gif|bmp)$")) {
            System.out.println("Error: Output file harus memiliki extension yang valid(.jpg, .png, etc).");
            outputPath = "";
```

```
            } else {
                // Validasi bahwa direktori tujuan ada dan dapat ditulis
                File outputFile = new File(outputPath);
                File parentDir = outputFile.getParentFile();
                if (parentDir != null && !parentDir.exists()) {
                    System.out.println("Warning: output directory yang dimaksud tidak tersedia.");
                }
            }
        } catch (Exception e) {
            System.out.println("Error saat membaca output path. Tolong coba lagi.");
            outputPath = "";
        }
    }

    try {
        // Tampilkan ringkasan parameter
        System.out.println("\nCompression Parameters:");
        System.out.println("Input Image: " + inputPath);
        System.out.println("Error Method: " + getErrorMethodName(errorMethod));
        System.out.println("Threshold: " + threshold);
        System.out.println("Minimum Block Size: " + minBlockSize);
        System.out.println("Output Image: " + outputPath);
        System.out.println("\nStarting compression...");

        // Buat dan jalankan kompresor
        QuadTreeCompression compressor = new QuadTreeCompression(inputPath, errorMethod, threshold, minBlockSize);
        compressor.compress();

        // Simpan hasil
        compressor.saveCompressedImage(outputPath);

        compressor.printStatistics();

    } catch (Exception e) {
        System.out.println("Sebuah erorr terjadi saat proses kompresi: " + e.getMessage());
    } finally {
        scanner.close();
    }
}

/**
 * Helper method untuk mendapatkan nama metode error
 */
private static String getErrorMethodName(int method) {
    return switch (method) {
        case 1 -> "Variance";
        case 2 -> "Mean Absolute Deviation (MAD)";
        case 3 -> "Max Pixel Difference";
```

```
        case 4 -> "Entropy";
        default -> "Unknown";
    };
}
}
```

Tangkapan layar input & ouput

Percobaan 1:

Input:

```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\ITB.jpg
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 1
Masukkan nilai threshold: 300
Masukkan minimum block size: 4
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\ITBAfter.jpg
```

Output:

```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\ITBAfter.jpg
Waktu Eksekusi: 65 ms
Kedalaman Pohon: 6
Jumlah Nodes: 869
Ukuran Gambar Asli: 9392 bytes
Ukuran Gambar Terkompresi: 4896 bytes
Persentase Kompresi: 47,87%
```

Percobaan 2:

Input:

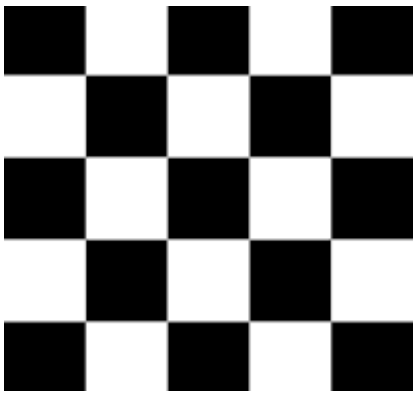
```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\blue.jpg
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 2
Masukkan nilai threshold: 35
Masukkan minimum block size: 8
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\blueAfter.jpg
```

Output:

```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\blueAfter.jpg
Waktu Eksekusi: 406 ms
Kedalaman Pohon: 7
Jumlah Nodes: 877
Ukuran Gambar Asli: 351019 bytes
Ukuran Gambar Terkompresi: 51596 bytes
Persentase Kompresi: 85,30%
```

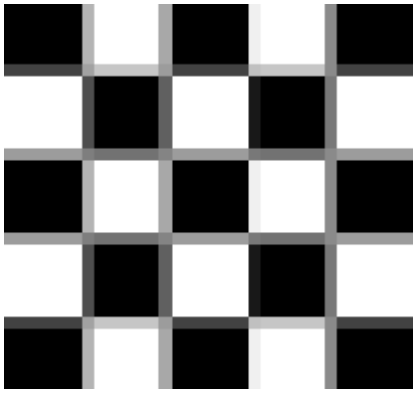
Percobaan 3:

Input:



```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\Checkboard.png
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 3
Masukkan nilai threshold: 10
Masukkan minimum block size: 4
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\CheckboardAfter.png
```

Output:



```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\CheckboardAfter.png
Waktu Eksekusi: 44 ms
Kedalaman Pohon: 6
Jumlah Nodes: 725
Ukuran Gambar Asli: 339 bytes
Ukuran Gambar Terkompresi: 1254 bytes
Persentase Kompresi: -269,91%
```

Percobaan 4:

Input:

```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\flower.jpg
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 4
Masukkan nilai threshold: 2
Masukkan minimum block size: 16
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\flowerAfter.jpg
```

Output:

```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\flowerAfter.jpg
Waktu Eksekusi: 515 ms
Kedalaman Pohon: 7
Jumlah Nodes: 5321
Ukuran Gambar Asli: 116048 bytes
Ukuran Gambar Terkompresi: 182390 bytes
Persentase Kompresi: -57,17%
PS C:\Users\user\Tucil2_13523155> java -cp bin QuadTreeCompression
```

Percobaan 5:

Input:

```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\Fullblue.png
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 4
Masukkan nilai threshold: 2
Masukkan minimum block size: 4
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\FullblueAfter.png
```

Output:

```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\FullblueAfter.png
Waktu Eksekusi: 18 ms
Kedalaman Pohon: 1
Jumlah Nodes: 1
Ukuran Gambar Asli: 143 bytes
Ukuran Gambar Terkompresi: 1090 bytes
Persentase Kompresi: -662,24%
```

Percobaan 6:

Input:


```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\kuning.png
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 1
Masukkan nilai threshold: 300
Masukkan minimum block size: 4
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\kuningAfter.png
```

Output:


```
Starting compression...
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\kuningAfter.png
Waktu Eksekusi: 20 ms
Kedalaman Pohon: 1
Jumlah Nodes: 1
Ukuran Gambar Asli: 144 bytes
Ukuran Gambar Terkompresi: 1192 bytes
Persentase Kompresi: -727,78%
```

Percobaan 7:

Input:

```
Image Compression using Quadtree
================================
Masukkan absolute path untuk input gambar: C:\Users\user\Tucil2_13523155\test\warnawarna.jpeg
Pilih error measurement method:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Masukkan Pilihanmu (1-4): 2
Masukkan nilai threshold: 50
Masukkan minimum block size: 8
Masukkan absolute path untuk output gambar terkompresi: C:\Users\user\Tucil2_13523155\test\warnawarnaAfter.jpeg
```

Output:

```
Compressed image saved to: C:\Users\user\Tucil2_13523155\test\warnawarnaAfter.jpeg
Waktu Eksekusi: 92 ms
Kedalaman Pohon: 6
Jumlah Nodes: 105
Ukuran Gambar Asli: 114055 bytes
Ukuran Gambar Terkompresi: 6632 bytes
Persentase Kompresi: 94,19%
```

Analisis Percobaan Algoritma

Analisis Kompleksitas Algoritma

Kompleksitas Waktu:

- Pembangunan Quadtree: O(n log n) dalam kasus rata-rata, di mana n adalah jumlah piksel
    - Kasus terbaik: O(1) - jika seluruh gambar berwarna seragam

    - Kasus terburuk: O(n) - jika setiap piksel harus menjadi leaf node terpisah
- Perhitungan Error: O(m) untuk setiap blok dengan m piksel
    - Variance, MAD: O(m)
    - Entropy: O(m + 256) ≈ O(m) untuk m yang besar
- Normalisasi Warna: O(n) - setiap piksel diproses sekali

Kompleksitas Ruang

- Struktur Quadtree: O(k) di mana k adalah jumlah node ($1 \leq k \leq n$)
    - Kasus terbaik: O(1) - satu node untuk seluruh gambar
    - Kasus terburuk: O(n) - satu node per piksel

Analisis Hasil Kompresi

Efektivitas Kompresi:

- Efektivitas kompresi sangat bergantung pada karakteristik gambar:
- Gambar dengan area warna seragam yang luas (seperti warna solid) dikompresi dengan sangat efisien
- Gambar dengan detail tinggi dan variasi warna (seperti foto) kurang terkompresi dengan baik

Parameter Optimal:

- Threshold: Parameter paling kritis yang mempengaruhi tradeoff antara kualitas dan kompresi
    - Nilai optimal berbeda untuk setiap metode error dan jenis gambar dari hasil percobaan, nilai yang optimal untuk setiap metode adalah:
      Variance: 30-100
      MAD: 5-20
      Max Pixel Difference: 10-40
      Entropy: 0.5-3.0
- Minimum Block Size: Membatasi pembagian berlebihan, mengurangi overhead

Perbandingan Metode Error:

- Variance: Baik untuk mendeteksi kontras warna
- MAD: Lebih toleran terhadap variasi gradual
- Max Pixel Difference: Sensitif terhadap outlier
- Entropy: Paling baik untuk kompleksitas visual yang dirasakan manusia

Tantangan dan Solusi:

- Ukuran File: Dalam implementasi awal, file hasil kompresi bisa lebih besar dari aslinya karena disimpan sebagai gambar reguler. Solusinya adalah dengan menyimpan struktur quadtree langsung dalam format biner kustom.
- Artefak Visual: Garis-garis pada batas blok dapat ditangani dengan teknik post-processing atau penggunaan threshold yang lebih rendah.

Kesimpulan

Kompresi gambar dengan metode Quadtree menggunakan pendekatan divide and conquer merupakan teknik kompresi yang efektif untuk gambar dengan area warna seragam yang luas. Implementasi yang telah dibuat berhasil mendemonstrasikan prinsip dasar kompresi quadtree dan menunjukkan bagaimana parameter seperti threshold, metode error, dan minimum block size mempengaruhi hasil kompresi.

Meskipun kompresi quadtree mungkin tidak selalu menghasilkan ukuran file yang lebih kecil dalam format gambar standar, implementasi penyimpanan struktur quadtree secara langsung menunjukkan potensi kompresi yang signifikan. Algoritma ini juga memiliki keunggulan dalam hal adaptasi terhadap konten gambar, di mana area dengan detail tinggi direpresentasikan dengan resolusi tinggi sementara area seragam dikompresi secara efisien.

Untuk pengembangan lebih lanjut, algoritma ini dapat ditingkatkan dengan teknik post-processing untuk mengurangi artefak visual, implementasi format kompresi kustom yang lebih efisien, dan optimasi kompleksitas algoritma untuk memproses gambar berukuran besar.

Lampiran

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	√	
2. Program berhasil dijalankan	√	
3. Program berhasil melakukan kompresi gambar sesuai dengan parameter yang ditentukan	√	
4. Mengimplementasi seluruh 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	√	

Pranala Ke Repository:

https://github.com/AbizzarG/Tucil2_13523155