LAPORAN TUGAS KECIL 2

IF2211 STRATEGI ALGORITMA

Kompresi Gambar Dengan Metode Quadtree

Disusun oleh:

Jethro Jens Norbert Simatupang – 13523081

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2025

DAFTAR ISI

BAB 1
PENDAHULUAN

1.1. Latar Belakang

Quadtree merupakan struktur data hierarkis yang umum digunakan dalam pengolahan gambar, khususnya untuk keperluan kompresi. Metode ini bekerja dengan membagi citra menjadi empat bagian secara rekursif berdasarkan keseragaman warna atau intensitas piksel menggunakan sistem warna RGB. Jika suatu bagian belum memenuhi tingkat keseragaman tertentu, maka proses pembagian akan dilanjutkan hingga mencapai ukuran minimum atau mencapai keseragaman.

Dalam implementasinya, Quadtree direpresentasikan sebagai node yang dapat memiliki hingga empat anak. Node daun mewakili area gambar yang seragam, sementara node internal menunjukkan area yang masih perlu dibagi. Setiap node menyimpan informasi seperti posisi, ukuran, dan rata-rata nilai warna. Pembagian gambar menjadi node-node ini merupakan salah satu contoh penerapan dari algoritma *divide and conquer*. Pendekatan ini efektif dalam mengurangi redundansi data dan sering digunakan dalam kompresi lossy karena dapat menurunkan ukuran file tanpa menghilangkan detail penting gambar.

1.2. Tujuan

Tujuan tugas kecil ini adalah:

a. Mengimplementasikan algoritma *divide and conquer* untuk melakukan kompresi gambar dengan metode quadtree
b. Menganalisis penerapan algoritma *divide and conquer* dalam kompresi gambar dengan metode quadtree

1.3. Teori Singkat

Algoritma *divide and conquer* adalah salah satu pendekatan pemrograman dan algoritma yang umum digunakan untuk menyelesaikan masalah kompleks dengan cara membaginya menjadi bagian-bagian yang lebih kecil. Algoritma ini dapat membantu dalam mengurangi kompleksitas algoritma dengan membuat program menjadi lebih modular.

Secara umum, proses algoritma *divide and conquer* melibatkan tiga tahap utama. Tahap pertama adalah *divide*, yaitu membagi masalah menjadi beberapa sub-masalah yang serupa dengan masalah awal, tetapi memiliki ukuran yang lebih kecil, idealnya seimbang satu sama lain. Selanjutnya, pada tahap *conquer*, setiap sub-masalah diselesaikan secara mandiri. Jika ukuran sub-masalah sudah cukup kecil, penyelesaiannya dapat dilakukan langsung. Namun jika masih cukup besar, sub-masalah tersebut akan diproses kembali menggunakan pendekatan yang sama secara rekursif. Terakhir, tahap *combine* dilakukan dengan cara menggabungkan solusi dari seluruh sub-masalah untuk mendapatkan solusi akhir dari permasalahan utama.

BAB 2
ALGORITMA

2.1. Notasi *Pseudocode*

a) public static void main(String[] args)

```
procedure main(input args: string[])
{Input}
input(scanner)
input(input)
inputFile ← File(inputPath)
input(outputPath)
format ← substring(outputPath, lastIndexOf(".", outputPath) + 1)
output ← BufferedImage(getWidth(input), getHeight(input), TYPE_INT_RGB)

{Variabel untuk output}
start ← System.nanoTime()
root ← compress(input, output, 0, 0, getWidth(input), getHeight(input),
0)
end ← System.nanoTime()
ImageIO.write(output, format, File(outputPath))
outFile ← File(outputPath)

{Output}
write("Waktu eksekusi: " + formatFloat((end - start) / 1e6, 2) + " ms")
write("Ukuran sebelum: " + div(length(inputFile), 1024) + " KB")
write("Ukuran sesudah: " + div(length(outFile), 1024) + " KB")
write("Persentase kompresi: " + formatFloat(100.0 * (1 -
(outFile.length() / inputFile.length())), 2) + "%")
write("Banyak simpul: " + nodeCount)
write("Kedalaman maksimum: " + maxTreeDepth)
```

b) static Node compress(BufferedImage input, BufferedImage output, int x, int y, int width, int height, int depth)

```
{Proses kompresi}
function compress(input: BufferedImage, output: BufferedImage, x:
integer, y: integer, width: integer, height: integer, depth: integer) →
Node
    error ← calculateError(input, x, y, width, height)

    {Basis: Jika luas blok sudah mencapai batas minimum atau sudah
homogen}
    if ((width * height) div 4 < minBlockSize) atau (height div 2 < 1)
atau (width div 2 < 1) atau (error < threshold) then
        avg ← averageColor(input, x, y, width, height)
        for i ← y to y + height - 1 do
            for j ← x to x + width - 1 do
                setRGB(output, j, i, getRGB(avg))
            endfor
        endfor

        nodeCount ← nodeCount + 1
        maxTreeDepth ← max(maxTreeDepth, depth)
        return Node(x, y, width, height, true)
    else
        {Rekursi: Pembagian blok jika belum mencapai minBlockSize dan
belum homogen}
        w2 ← width div 2
        h2 ← height div 2
        node ← Node(x, y, width, height, false)
        node.children ← array[4]
```

```
        {Kompresi masing-masing blok}
        node.children[0] ← compress(input, output, x, y, w2, h2, depth +
1)
        node.children[1] ← compress(input, output, x + w2, y, width - w2,
h2, depth + 1)
        node.children[2] ← compress(input, output, x, y + h2, w2, height
- h2, depth + 1)
        node.children[3] ← compress(input, output, x + w2, y + h2, width
- w2, height - h2, depth + 1)

        return node
    endif
```

c) public static BufferedImage getInputImage(Scanner scanner)

```
{Input alamat absolut gambar yang akan dikompresi}
function getInputImage(scanner: Scanner) → BufferedImage
    img ← null

    while true do
        write("Alamat absolut gambar: ")
        input(path)
        file ← File(path)

        if file.exists() = false then
            write("File tidak ditemukan. Silakan coba lagi.")
            continue
        else
            try
                img ← readImage(file)
                if img = null then
                    write("File tidak bisa dibaca sebagai gambar. Silakan
coba lagi.")
                    continue
                else
                    inputPath ← path
                    return img
                endif
            catch IOException e
                write("Terjadi kesalahan saat membaca file: " +
getMessage(e))
            endtry
        endif
    endwhile
```

d) public static void getMethodThresholdBlockSize(Scanner scanner, BufferedImage inputImage)

```
{Input pilihan metode perhitungan variansi, ambang batas, dan ukuran blok
minimum}
procedure getMethodThresholdBlockSize(input scanner: Scanner, input
inputImage: BufferedImage)
    write("PILIH METODE PERHITUNGAN VARIANSI")
    write("1. Variance")
    write("2. Mean Absolute Deviation (MAD)")
    write("3. Max Pixel Difference")
    write("4. Entropy")

    { Validasi metode }
    while true do
        write("Pilihan metode: ")
        input(line)
        try
            m ← konversiKeInteger(line)
```

```
            if m ≥ 1 and m ≤ 4 then
                method ← m
                break
            else
                write("Input tidak valid. Masukkan angka antara 1 hingga
4.")
            endif
        catch
            write("Input tidak valid. Harap masukkan angka.")
        endtry
    endwhile

    { Validasi threshold }
    while true do
        write("Threshold: ")
        input(line)
        try
            t ← konversiKeReal(line)

            if method = 4 then
                isValid ← t ≥ 0.0 and t ≤ 8.0
            else
                isValid ← t ≥ 0.0 and t ≤ 16256.25
            endif

            if isValid then
                threshold ← t
                break
            else
                if method = 4 then
                    write("Threshold untuk Entropy harus antara 0.0 dan
8.0.")
                else
                    write("Threshold harus antara 0.0 dan 16256.25.")
                endif
            endif
        catch
            write("Input tidak valid. Harap masukkan angka untuk
threshold.")
        endtry
    endwhile

    { Validasi minimum block size }
    maxBlockSize ← getWidth(inputImage) * getHeight(inputImage)
    while true do
        write("Minimum block size: ")
        input(line)
        try
            size ← konversiKeInteger(line)
            if size ≥ 1 and size ≤ maxBlockSize then
                minBlockSize ← size
                break
            else
                write("Ukuran blok harus antara 1 dan " + maxBlockSize +
".")
            endif
        catch
            write("Input tidak valid. Harap masukkan angka untuk ukuran
blok.")
        endtry
    endwhile
```

e) public static String getOutputPath(Scanner scanner, File inputFile)

```
{Input alamat absolut gambar hasil kompresi}
function getOutputPath(scanner: Scanner, inputFile: BufferedImage) →
string
    while true do
        write("Alamat output: ")
        input(out)
        outputFile ← File(out)

        try
            if getCanonicalPath(outputFile) = getCanonicalPath(inputFile)
then
                write("Output tidak boleh sama dengan input.")
                continue
            endif
        catch
            write("Gagal memverifikasi path output. Coba lagi.")
            continue
        endtry

        if not contains(out, ".") then
            write("Path output harus menyertakan ekstensi (misalnya:
hasil.jpg).")
            continue
        endif

        parentDir ← getParentFile(outputFile)
        if parentDir ≠ null and not exists(parentDir) then
            write("Direktori tidak ditemukan: " +
getAbsolutePath(parentDir))
            continue
        endif

        ext ← toLowerCase(substring(out, lastIndexOf(out, ".") + 1))
        supported ← false
        for format in getWriterFormatNames(ImageIO) do
            if equalsIgnoreCase(format, ext) then
                supported ← true
                break
            endif
        endfor

        if not supported then
            write("Format file tidak didukung: " + ext)
            continue
        endif

        return out
    endwhile
```

f) public static Color averageColor(BufferedImage img, int x, int y, int width, int height)

```
{Perhitungan rata-rata warna untuk normalisasi warna}
function averageColor(img: BufferedImage, x: integer, y: integer, width:
integer, height: integer) → Color
    r ← 0
    g ← 0
    b ← 0
```

```
    count ← width * height

    for i ← y to y + height - 1 do
        for j ← x to x + width - 1 do
            c ← Color(getRGB(img, j, i))
            r ← r + getRed(c)
            g ← g + getGreen(c)
            b ← b + getBlue(c)
        endfor
```

g) public static double calculateError(BufferedImage img, int x, int y, int width, int height)

```
function calculateError(img: BufferedImage, x: integer, y: integer,
width: integer, height: integer) → Real
    N ← width * height
    sum ← array[0, 0, 0]

    {Penghitungan jumlah total RGB dari semua piksel dalam blok}
    for i ← y to y + height - 1 do
        for j ← x to x + width - 1 do
            c ← Color(getRGB(img, j, i))
            sum[0] ← sum[0] + getRed(c)
            sum[1] ← sum[1] + getGreen(c)
            sum[2] ← sum[2] + getBlue(c)
        endfor
    endfor

    {Perhitungan rata-rata warna untuk RGB}
    mean ← array[sum[0] div N, sum[1] div N, sum[2] div N]

    {Perhitungan error berdasarkan metode yang dipilih}
    if ImageCompressor.method = 1 then
        return calculateVariance(img, x, y, width, height, mean)
    else
    if ImageCompressor.method = 2 then
        return calculateMAD(img, x, y, width, height, mean)
    else
    if ImageCompressor.method = 3 then
        return calculateMaxDiff(img, x, y, width, height)
    else
    if ImageCompressor.method = 4 then
        return calculateEntropy(img, x, y, width, height)
    else
        return 0.0
    endif
    endif
    endif
    endif
```

h) private static double calculateVariance(BufferedImage img, int x, int y, int width, int height, double[] mean)

```
{Metode perhitungan error: Variance}
function calculateVariance(img: BufferedImage, x: integer, y: integer,
width: integer, height: integer, mean: array [] of integer) → Real
    N ← width * height
    error ← 0

    for c ← 0 to 2 do
        var ← 0
```

```
        for i ← y to y + height - 1 do
            for j ← x to x + width - 1 do
                color ← Color(getRGB(img, j, i))
                if c = 0 then
                    value ← getRed(color)
                else
                if c = 1 then
                    value ← getGreen(color)
                else
                    value ← getBlue(color)
                endif
                endif
                var ← var + pow(value - mean[c], 2)
            endfor
        endfor
        error ← error + (var div N) {σ_c² = (1 / N) * Σ_{i = 1 to N}
(P_{i,c} - μ_c)²}
    endfor

    return error div 3 {σ_RGB² = (σ_R² + σ_G² + σ_B²) / 3}
```

i) private static double calculateMAD(BufferedImage img, int x, int y, int width, int height, double[] mean)

```
{Metode perhitungan error: MAD}
function calculateMAD(img: BufferedImage, x: integer, y: integer, width:
integer, height: integer, mean: array [] of double) → Real
    N ← width * height
    error ← 0

    for c ← 0 to 2 do
        mad ← 0
        for i ← y to y + height - 1 do
            for j ← x to x + width - 1 do
                color ← Color(getRGB(img, j, i))
                if c = 0 then
                    value ← getRed(color)
                else
                if c = 1 then
                    value ← getGreen(color)
                else
                    value ← getBlue(color)
                endif
                endif
                mad ← mad + abs(value - mean[c])
            endfor
        endfor
        error ← error + (mad div N) {MAD_c = (1 / N) * Σ_{i = 1 to N}
(|P_{i,c} - μ_c|)}
    endfor

    return error div 3 {MAD_RGB = (MAD_R + MAD_G + MAD_B) / 3}
```

j) private static double calculateMaxDiff(BufferedImage img, int x, int y, int width, int height)

```
{Metode perhitungan error: Max Pixel Difference}
function calculateMaxDiff(img: BufferedImage, x: integer, y: integer,
width: integer, height: integer) → Real
    error ← 0
```

```
    for c ← 0 to 2 do
        min ← 255
        max ← 0
        for i ← y to y + height - 1 do
            for j ← x to x + width - 1 do
                color ← Color(getRGB(img, j, i))
                if c = 0 then
                    value ← getRed(color)
                else
                if c = 1 then
                    value ← getGreen(color)
                else
                    value ← getBlue(color)
                endif
                endif
                min ← min(min, value)
                max ← max(max, value)
            endfor
        endfor
        error ← error + (max - min) {D_c = max(P_{i,c}) - min(P_{i,c})}
    endfor

    return error div 3 {D_RGB = (D_R + D_G + D_B) / 3}
```

k) private static double calculateEntropy(BufferedImage img, int x, int y, int width, int height)

```
{Menghitung nilai Entropi dari gambar pada area tertentu}
function calculateEntropy(img, x, y, width, height) → Real
    N ← width * height
    error ← 0

    for c ← 0 to 2 do
        hist ← array of 256 integer initialized with 0

        for i ← y to y + height - 1 do
            for j ← x to x + width - 1 do
                color ← Color(getRGB(img, j, i))
                if c = 0 then
                    value ← getRed(color)
                else
                if c = 1 then
                    value ← getGreen(color)
                else
                    value ← getBlue(color)
                endif
                endif
                hist[value] ← hist[value] + 1
            endfor
        endfor

        H ← 0
        for each h in hist do
            if h > 0 then
                p ← h div N
                H ← H - (p * (log(p) div log(2)))
            endif
        endfor

        error ← error + H {H_c = - Σ (P_c(i) log_2(P_c(i)))}
    endfor

    return error div 3 {H RGB = (H R + H G + H B) / 3}
```

l) `public Node(int x, int y, int width, int height, boolean isLeaf)`

```
class Node
    {Struktur data untuk node}
    x, y, width, height: integer
    isLeaf: boolean
    children: array[] of Node

    procedure Node(input x: integer, input y: integer, input width:
integer, input height: integer, input isLeaf: boolean)
        this.x ← x
        this.y ← y
        this.width ← width
        this.height ← height
        this.isLeaf ← isLeaf
```

2.2. Penjelasan Algoritma *Divide and Conquer*

Algoritma yang dibuat menggunakan algoritma *divide and conquer* dengan membagi gambar menjadi node-node dan melakukan kompresi terhadap node-node tersebut. Berikut adalah penjelasan langkah-langkahnya:

a) Inisialisasi dan Persiapan Data (Pra-proses)
Sebelum masuk ke algoritma utama, program melakukan proses berikut:
- Membaca gambar input dalam format RGB dan memuatnya dalam bentuk BufferedImage.
- Membaca masukan parameter kompresi dari pengguna, yaitu pilihan metode variansi, ambang batas, dan ukuran minimum blok piksel. *Nilai maksimum ambang batas untuk metode Variance, MAD, dan Max Pixel Difference* adalah 16.256,25, yang diperoleh dari perhitungan $Var_{max} = (255 - \left(\frac{0+255}{2}\right))^2 = 16256{,}25$). Sementara untuk metode *Entropy*, nilai maksimum ambang batas adalah 8,0, berdasarkan perhitungan $H_{RGBmax} = \frac{H_{Rmax}+H_{Gmax}+H_{Bmax}}{3} = \frac{log_2 256+log_2 256+log_2 256}{3} = 8$
- Mempersiapkan struktur data Node untuk menyusun hasil kompresi dalam bentuk quadtree.

b) Perhitungan Error
Sebelum melakukan pembagian suatu blok, program memeriksa kehomogenan blok dengan menghitung nilai *error* dari blok tersebut berdasarkan metode yang dipilih. Proses ini dilakukan pada fungsi calculateError() yang terbagi menjadi empat metode, yaitu:
- Variance: menghitung sebaran nilai RGB
- MAD: menghitung deviasi absolut rata-rata dari nilai RGB
- Max Diff: selisih maksimum nilai RGB
- Entropy: ukuran ketidakteraturan nilai RGB dalam blok

c) Pembagian Blok (*Divide*)
Pada tahap ini, program membagi blok secara rekursif menjadi 4 sub-blok. Proses ini merupakan tahapan *divide* pada algoritma *divide and conquer* yang membagi gambar menjadi blok-blok kecil yang akan diproses satu per satu. Adapun syarat pembagian blok adalah error blok masih lebih besar daripada threshold dan ukuran

sub-blok setelah dibagi tidak lebih kecil dari minBlockSize. Jika kedua kondisi tersebut terpenuhi, maka fungsi compress() akan membagi blok menjadi empat kuadran (kiri-atas, kanan-atas, kiri-bawah, kanan-bawah). Setiap kuadran akan diproses kembali secara rekursif menggunakan compress().

d) Normalisasi Warna (*Conquer*)
Proses ini merupakan bagian basis dari proses rekursif yang akan dieksekusi jika blok sudah tidak akan dibagi lagi (karena sudah homogen atau terlalu kecil). Proses ini merupakan tahapan *conquer* pada algoritma *divide and conquer* yang memproses setiap blok secara mandiri. Blok-blok yang diproses di sini akan dianggap sebagai daun (leaf). Adapun proses yang dilakukan di sini adalah program akan menghitung rata-rata warna RGB dalam blok (averageColor()). Semua piksel dalam blok tersebut akan diset ke warna rata-rata (dioutputkan ke outputImage) dan node yang sedang diproses akan ditandai sebagai isLeaf = true.

e) Rekonstruksi dan Output (*Combine*)
Setelah semua blok selesai diproses, struktur quadtree akan digunakan untuk menyusun ulang gambar ke dalam bentuk terkompresi (outputImage). Proses ini merupakan tahapan *combine* pada algoritma *divide and conquer* yang menyatukan setiap node yang sudah diproses secara mandiri. Gambar hasil kompresi disimpan menggunakan ImageIO.write(). Selain itu, informasi tambahan kompresi akan ditampilkan, yang meliputi waktu eksekusi, ukuran file asli, ukuran file hasil, persentase kompresi, kedalaman pohon, dan jumlah node pada quadtree.

BAB 3
SOURCE CODE

3.1 Main.java

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

public class Main {
    public static int method;
    public static double threshold;
    public static int minBlockSize;
    public static String inputPath;

    public static int nodeCount = 0;
    public static int maxTreeDepth = 0;

    public static void main(String[] args) throws Exception {
        // Input
        Scanner scanner = new Scanner(System.in);
        BufferedImage input = IOHandler.getInputImage(scanner);
        IOHandler.getMethodThresholdBlockSize(scanner, input);
        File inputFile = new File(inputPath);
        String outputPath = IOHandler.getOutputPath(scanner, inputFile);
        String format = outputPath.substring(outputPath.lastIndexOf('.') + 1);
        BufferedImage output = new BufferedImage(input.getWidth(), input.getHeight(), BufferedImage.TYPE_INT_RGB);

        // Variabel untuk output
        long start = System.nanoTime();
        Node root = compress(input, output, 0, 0, input.getWidth(), input.getHeight(), 0);
        long end = System.nanoTime();
        ImageIO.write(output, format, new File(outputPath));
        File outFile = new File(outputPath);

        // Output
        System.out.printf("Waktu ekseskusi: %.2f ms\n", (end - start) / 1e6);
        System.out.printf("Ukuran sebelum: %d KB\n", inputFile.length() / 1024);
        System.out.printf("Ukuran sesudah: %d KB\n", outFile.length() / 1024);
        System.out.printf("Persentase kompresi: %.2f%%\n", 100.0 * (1 - (double) outFile.length() / inputFile.length()));
        System.out.println("Banyak simpul: " + nodeCount);
        System.out.println("Kedalaman maksimum: " + maxTreeDepth);
    }

    // Proses kompresi
    public static Node compress(BufferedImage input, BufferedImage output, int x, int y, int width, int height, int depth) {
        double error = ImageUtils.calculateError(input, x, y, width, height);

        // Basis: Jika luas blok sudah mencapai batas minimum atau sudah homogen
        if (((width * height) / 4 < minBlockSize) || height / 2 < 1 || width / 2 < 1 || error < threshold) {
            // Normalisasi warna
            Color avg = ImageUtils.averageColor(input, x, y, width, height);
            for (int i = y; i < y + height; i++) {
                for (int j = x; j < x + width; j++) {
                    output.setRGB(j, i, avg.getRGB());
                }
            }
            nodeCount++;
            maxTreeDepth = Math.max(maxTreeDepth, depth);
            return new Node(x, y, width, height, true);
        }

        // Rekursi: Pembagian blok jika melum mencapai minBlockSize dan belum homogen
        int w2 = width / 2;
        int h2 = height / 2;
        Node node = new Node(x, y, width, height, false);
        node.children = new Node[4];

```

```
        // Kompresi masing-masing blok
        node.children[0] = compress(input, output, x, y, w2, h2, depth + 1);
        node.children[1] = compress(input, output, x + w2, y, width - w2, h2, depth + 1);
        node.children[2] = compress(input, output, x, y + h2, w2, height - h2, depth + 1);
        node.children[3] = compress(input, output, x + w2, y + h2, width - w2, height - h2, depth + 1);

        return node;
    }
}
```

3.2 IOHandler.java

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

public class IOHandler {

    // Input alamat absolut gambar yang akan dikompresi
    public static BufferedImage getInputImage(Scanner scanner) {
        BufferedImage img = null;

        while (true) {
            System.out.print("Alamat absolut gambar: ");
            String path = scanner.nextLine().trim().replace("\"", "");
            File file = new File(path);

            if (!file.isAbsolute()) {
                System.out.println("Harap masukkan path absolut (misalnya: C:\\Users\\Acer\\Documents\\input.png).");
                continue;
            }

            if (!file.exists()) {
                System.out.println("File tidak ditemukan. Silakan coba lagi.");
                continue;
            }

            try {
                img = ImageIO.read(file);
                if (img == null) {
                    System.out.println("File tidak bisa dibaca sebagai gambar. Silakan coba lagi.");
                    continue;
                }
                ImageCompressor.inputPath = path;
                return img;
            } catch (IOException e) {
                System.out.println("Terjadi kesalahan saat membaca file: " + e.getMessage());
            }
        }
    }

    // Input pilihan metode perhitungan variansi, ambang batas, dan ukuran blok minimum
    public static void getMethodThresholdBlockSize(Scanner scanner, BufferedImage inputImage) {
        System.out.println("PILIH METODE PERHITUNGAN VARIANSI");
        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");

        // Validasi metode
        while (true) {
            System.out.print("Pilihan metode: ");
            try {
                int m = Integer.parseInt(scanner.nextLine());
                if (m >= 1 && m <= 4) {
                    ImageCompressor.method = m;
                    break;
                } else {
                    System.out.println("Input tidak valid. Masukkan angka antara 1 hingga 4.");
                }
            } catch (NumberFormatException e) {
                System.out.println("Input tidak valid. Harap masukkan angka.");
            }
        }
```

```java

    // Validasi threshold
    while (true) {
        System.out.print("Threshold: ");
        try {
            double t = Double.parseDouble(scanner.nextLine());

            boolean isValid = switch (ImageCompressor.method) {
                case 4 -> t >= 0.0 && t <= 8.0;
                default -> t >= 0.0 && t <= 16256.25;
            };

            if (isValid) {
                ImageCompressor.threshold = t;
                break;
            } else {
                System.out.println(ImageCompressor.method == 4
                    ? "Threshold untuk Entropy harus antara 0.0 dan 8.0."
                    : "Threshold harus antara 0.0 dan 16256.25.");
            }
        } catch (NumberFormatException e) {
            System.out.println("Input tidak valid. Harap masukkan angka untuk threshold.");
        }
    }

    // Validasi minimum block size
    int maxBlockSize = inputImage.getWidth() * inputImage.getHeight();
    while (true) {
        System.out.print("Minimum block size: ");
        try {
            int size = Integer.parseInt(scanner.nextLine());
            if (size >= 1 && size <= maxBlockSize) {
                ImageCompressor.minBlockSize = size;
                break;
            } else {
                System.out.println("Ukuran blok harus antara 1 dan " + maxBlockSize + ".");
            }
        } catch (NumberFormatException e) {
            System.out.println("Input tidak valid. Harap masukkan angka untuk ukuran blok.");
        }
    }
}

// Input alamat absolut gambar hasil kompresi
public static String getOutputPath(Scanner scanner, File inputFile) {
    while (true) {
        System.out.print("Alamat output: ");
        String out = scanner.nextLine().trim().replace("\"", "");
        File outputFile = new File(out);

        if (!outputFile.isAbsolute()) {
            System.out.println("Harap masukkan path absolut (misalnya: C:\\Users\\Acer\\Documents\\output.png).");
            continue;
        }

        try {
            if (outputFile.getCanonicalPath().equals(inputFile.getCanonicalPath())) {
                System.out.println("Alamat output tidak boleh sama dengan alamat input.");
                continue;
            }
        } catch (IOException e) {
            System.out.println("Gagal memverifikasi path output. Coba lagi.");
            continue;
        }

        if (!out.contains(".")) {
            System.out.println("Path output harus menyertakan ekstensi.");
            continue;
        }

        File parentDir = outputFile.getParentFile();
        if (parentDir != null && !parentDir.exists()) {
            System.out.println("Direktori tidak ditemukan: " + parentDir.getAbsolutePath());
            continue;
        }

        String ext = out.substring(out.lastIndexOf('.') + 1).toLowerCase();
        boolean supported = false;
        for (String format : ImageIO.getWriterFormatNames()) {
            if (format.equalsIgnoreCase(ext)) {
                supported = true;
                break;
            }
        }
```

```
1
2               String ext = out.substring(out.lastIndexOf('.') + 1).toLowerCase();
3               boolean supported = false;
4               for (String format : ImageIO.getWriterFormatNames()) {
5                   if (format.equalsIgnoreCase(ext)) {
6                       supported = true;
7                       break;
8                   }
9               }
10
11              if (!supported) {
12                  System.out.println("Format file tidak didukung: " + ext);
13                  continue;
14              }
15
16              return out;
17          }
18      }
19  }
20
```

3.3 ImageUtils.java

```
1  import java.awt.Color;
2  import java.awt.image.BufferedImage;
3
4  public class ImageUtils {
5
6      // Perhitungan rata-rata warna untuk normalisasi warna
7      public static Color averageColor(BufferedImage img, int x, int y, int width, int height) {
8          long r = 0, g = 0, b = 0;
9          int count = width * height;
10
11         for (int i = y; i < y + height; i++) {
12             for (int j = x; j < x + width; j++) {
13                 Color c = new Color(img.getRGB(j, i));
14                 r += c.getRed();
15                 g += c.getGreen();
16                 b += c.getBlue();
17             }
18         }
19
20         return new Color((int)(r / count), (int)(g / count), (int)(b / count));
21     }
22
23     // Perhitungan error
24     public static double calculateError(BufferedImage img, int x, int y, int width, int height) {
25         int N = width * height;
26         double[] sum = new double[3];
27
28         // Penghitungan jumlah total RGB dari semua piksel dalam blok
29         for (int i = y; i < y + height; i++) {
30             for (int j = x; j < x + width; j++) {
31                 Color c = new Color(img.getRGB(j, i));
32                 sum[0] += c.getRed();
33                 sum[1] += c.getGreen();
34                 sum[2] += c.getBlue();
35             }
36         }
37
38         // Perhitungan rata-rata warna untuk RGB
39         double[] mean = { sum[0] / N, sum[1] / N, sum[2] / N };
40
41         // Perhitungan error berdasarkan metode yang dipilih
42         return switch (ImageCompressor.method) {
43             case 1 -> calculateVariance(img, x, y, width, height, mean);
44             case 2 -> calculateMAD(img, x, y, width, height, mean);
45             case 3 -> calculateMaxDiff(img, x, y, width, height);
46             case 4 -> calculateEntropy(img, x, y, width, height);
47             default -> 0.0;
48         };
49     }
50
```

```java
// Metode perhitungan error: Variance
private static double calculateVariance(BufferedImage img, int x, int y, int width, int height, double[] mean) {
    int N = width * height;
    double error = 0;

    for (int c = 0; c < 3; c++) {
        double var = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(img.getRGB(j, i));
                int value = (c == 0) ? color.getRed() : (c == 1) ? color.getGreen() : color.getBlue();
                var += Math.pow(value - mean[c], 2);
            }
        }
        error += var / N; // σ_c² = (1 / N) * Σ_{i = 1 to N} (P_{i,c} - μ_c)²
    }

    return error / 3; // σ_RGB² = (σ_R² + σ_G² + σ_B²) / 3
}

// Metode perhitungan error: MAD
private static double calculateMAD(BufferedImage img, int x, int y, int width, int height, double[] mean) {
    int N = width * height;
    double error = 0;

    for (int c = 0; c < 3; c++) {
        double mad = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(img.getRGB(j, i));
                int value = (c == 0) ? color.getRed() : (c == 1) ? color.getGreen() : color.getBlue();
                mad += Math.abs(value - mean[c]);
            }
        }
        error += mad / N; // MAD_c = (1 / N) * Σ_{i = 1 to N} (|P_{i,c} - μ_c|)
    }

    return error / 3; // MAD_RGB = (MAD_R + MAD_G + MAD_B) / 3
}

// Metode perhitungan error: Max Pixel Difference
private static double calculateMaxDiff(BufferedImage img, int x, int y, int width, int height) {
    double error = 0;

    for (int c = 0; c < 3; c++) {
        int min = 255, max = 0;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(img.getRGB(j, i));
                int value = (c == 0) ? color.getRed() : (c == 1) ? color.getGreen() : color.getBlue();
                min = Math.min(min, value);
                max = Math.max(max, value);
            }
        }
        error += max - min; // D_c = max(P_{i,c}) - min(P_{i,c})
    }

    return error / 3; // D_RGB = (D_R + D_G + D_B) / 3
}

// Metode perhitungan error: Entropy
private static double calculateEntropy(BufferedImage img, int x, int y, int width, int height) {
    int N = width * height;
    double error = 0;

    for (int c = 0; c < 3; c++) {
        int[] hist = new int[256];
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                Color color = new Color(img.getRGB(j, i));
                int value = (c == 0) ? color.getRed() : (c == 1) ? color.getGreen() : color.getBlue();
                hist[value]++;
            }
        }

```

```
            double H = 0;
            for (int h : hist) {
                if (h > 0) {
                    double p = h / (double) N;
                    H -= p * (Math.log(p) / Math.log(2));
                }
            }

            error += H; // H_c = - Σ_{i = 1 to N} (P_c(i) log_2(P_c(i)))
        }

        return error / 3; // H_RGB = (H_R + H_G + H_B) / 3
    }
}

```

3.4 Node.java

```
public class Node {
    // Struktur data untuk node
    int x, y, width, height;
    boolean isLeaf;
    Node[] children;

    public Node(int x, int y, int width, int height, boolean isLeaf) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.isLeaf = isLeaf;
    }
}

```

BAB 4
EKSPERIMEN

4.1. Test Case 1

Input:

Output:

```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input1.png
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 1
Threshold: 100
Minimum block size: 4
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output1.png
Waktu eksekusi: 285,96 ms
Ukuran gambar sebelum: 152 KB
Ukuran gambar setelah: 36 KB
Persentase kompresi: 75,94%
Kedalaman pohon: 9
Jumlah simpul: 8573
```

4.2. Test Case 2

Input:

Output:

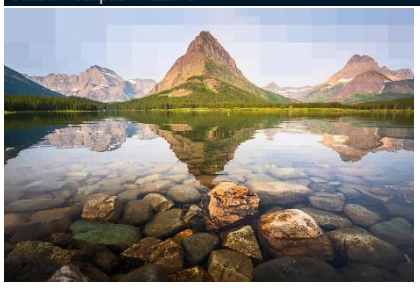
```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input2.jpg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 2
Threshold: 15
Minimum block size: 1
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output2.jpg
Waktu eksekusi: 13255,32 ms
Ukuran gambar sebelum: 1488 KB
Ukuran gambar setelah: 887 KB
Persentase kompresi: 40,34%
Kedalaman pohon: 13
Jumlah simpul: 429673
```

4.3. Test Case 3

Input:

Output:

```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input3.jpg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 3
Threshold: 30
Minimum block size: 1
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output3.jpg
Waktu eksekusi: 2958,60 ms
Ukuran gambar sebelum: 381 KB
Ukuran gambar setelah: 296 KB
Persentase kompresi: 22,22%
Kedalaman pohon: 12
Jumlah simpul: 422873
```



4.4. Test Case 4

Input:

Output:

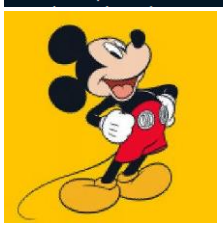
```
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output4.jpg
Waktu eksekusi: 10153,35 ms
Ukuran gambar sebelum: 994 KB
Ukuran gambar setelah: 553 KB
Persentase kompresi: 44,39%
Kedalaman pohon: 12
Jumlah simpul: 4559789
```

4.5. Test Case 5

Input:

Output:

```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input5.jpeg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 1
Threshold: 20
Minimum block size: 2
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output5.jpeg
Waktu eksekusi: 175,53 ms
Ukuran gambar sebelum: 28 KB
Ukuran gambar setelah: 14 KB
Persentase kompresi: 49,70%
Kedalaman pohon: 9
Jumlah simpul: 8809
```



4.6. Test Case 6

Input:

Output:

```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input6.jpg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 1
Threshold: 200
Minimum block size: 4
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output6.jpg
Waktu eksekusi: 121,07 ms
Ukuran gambar sebelum: 8 KB
Ukuran gambar setelah: 8 KB
Persentase kompresi: 5,62%
Kedalaman pohon: 8
Jumlah simpul: 2905
```

4.7. Test Case 7

Input:

Output:

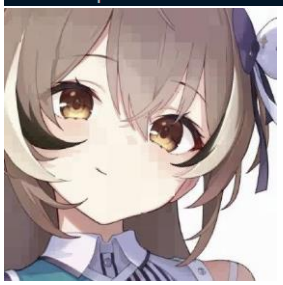
```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input7.jpg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 1
Threshold: 300
Minimum block size: 2
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output7.jpg
Waktu eksekusi: 441,34 ms
Ukuran gambar sebelum: 271 KB
Ukuran gambar setelah: 29 KB
Persentase kompresi: 89,30%
Kedalaman pohon: 10
Jumlah simpul: 11261
```

4.8. Test Case 8

Input:

Output:

```
Alamat absolut gambar: C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Input\input8.jpg
PILIH METODE PERHITUNGAN VARIANSI
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilihan metode perhitungan error: 2
Threshold: 5
Minimum block size: 1
Alamat absolut output (dengan extension): C:\Users\Acer\Documents\Kuliah\Semester 4\Strategi Algoritma\Tugas Kecil 2\Output\output8.jpg
Waktu eksekusi: 255,79 ms
Ukuran gambar sebelum: 32 KB
Ukuran gambar setelah: 25 KB
Persentase kompresi: 21,91%
Kedalaman pohon: 10
Jumlah simpul: 46937
```



BAB 5
ANALISIS

5.1 Analisis rekursi

Pada algoritma yang telah dibuat, fungsi compress memanggil dirinya sendiri 4 kali, masing-masing pada setengah lebar dan setengah tinggi gambar:

a) Jumlah pemanggilan rekursif = 4, maka a = 4
b) Setiap pemanggilan bekerja pada submasalah ukuran ½ dari dimensi → ukuran area jadi n/2 (untuk n = area gambar) → b = 2
c) Di luar pemanggilan rekursif, fungsi compress melakukan:
   - Hitung warna rata-rata
   - Hitung error
   - Set warna piksel

5.2 Teorema Master

Bentuk umum Teorema Master:

$T(n) = a\,T\left(\frac{n}{b}\right) + cn^d$

Dengan:

a) a: jumlah pemanggilan rekursif (submasalah)
b) b: faktor pembagi ukuran
c) cn^d: pekerjaan non-rekursif di setiap level

Dari analisis rekursi di atas, dapat kita identifikasikan:

a) $a = 4$ → karena rekursi dilakukan kepada keempat blok (4 pemanggilan compress)
b) $b = 2$ → karena masing-masing blok dibagi setengah ukurannya (lebar dan tingginya)
c) $f(n) = cn^d$ = pekerjaan per level, yaitu membaca piksel O(n^2), tetapi satu blok berukuran
   n × n, jadi
   $f(n) = \Theta(n^2) \rightarrow d = 2$

Berdasarkan perbandingan $a = b^d$, dapat dihitung kompleksitas algoritma:
$T(n) = O(n^d \log n) = O(n^2 \log n)$

5.3 Analisis Singkat

Berdasarkan hasil uji coba, dapat diketahui bahwa metode kompresi gambar berbasis quadtree ini memiliki keunggulan dalam menangani gambar-gambar dengan karakteristik warna yang homogen dan minim detail. Pada gambar seperti ilustrasi sederhana atau gambar dengan latar polos, algoritma ini mampu secara efisien membagi blok-blok gambar tanpa perlu rekursi yang dalam, menghasilkan rasio kompresi yang tinggi tanpa mengorbankan kualitas visual secara signifikan. Namun, pendekatan ini menjadi kurang efektif ketika diterapkan pada gambar dengan resolusi tinggi yang memiliki latar belakang kompleks atau penuh detail, seperti pemandangan alam atau gambar dengan banyak tekstur. Dalam kasus tersebut, algoritma cenderung terus membagi blok hingga ukuran minimum, menghasilkan struktur pohon yang sangat dalam dan waktu eksekusi akan cenderung lama. Citra hasil kompresi juga bisa terlihat "kotak-kotak" atau pixelated

jika ukuran blok minimumnya besar (melebihi ukuran detail gambar) atau jika ambang batas terlalu besar. Selain itu, metode ini juga kurang cocok untuk gambar berukuran kecil, karena ukuran blok minimum yang telah ditentukan bisa jadi sudah mendekati atau melebihi dimensi gambar itu sendiri. Hal ini menyebabkan pembagian blok menjadi sangat terbatas dan efektivitas kompresi menurun drastis. Dengan demikian, metode quadtree lebih ideal digunakan untuk gambar-gambar dengan variasi warna rendah dan pola visual yang tidak terlalu kompleks.

LAMPIRAN

1. **Repository GitHub**
   Berikut adalah pranala ke repository GitHub yang berisi kode sumber dan dokumentasi proyek ini:
   https://github.com/JethroJNS/Tucil2_13523081.git
2. **Tabel Evaluasi**

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	✓	