TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA

Benedictus Nelson
13523150

Dosen Pengampu :
Dr. Nur Ula Maulidevi, S.T, M.Sc.
Dr. Ir. Rinaldi Munir, M.T.
Monterico Adrrian, S.T, M.T.

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2025

DAFTAR ISI

BAB 1 ALGORITMA

1.1 Main Idea

Tujuan Utama Program

Melakukan kompresi gambar berbasis algoritma Divide and Conquer dengan memanfaatkan struktur data Quadtree. Program bekerja dengan cara membagi gambar ke dalam blok-blok kecil berdasarkan nilai keseragaman warna, kemudian merepresentasikan gambar dalam bentuk pohon Quadtree.

1.2 Langkah Terinci

Parse Input

Program menerima input dari user berupa:

- Alamat file gambar asli
- Metode perhitungan error (1 = Variance, 2 = MAD, 3 = MaxDiff, 4 = Entropy, 5 = SSIM)
- Threshold error
- Ukuran blok minimum
- Target kompresi (opsional)
- Alamat file untuk menyimpan gambar hasil kompresi
- Alamat file untuk menyimpan GIF visualisasi (opsional)

Persiapan Data Struktural

- originalImage = Gambar awal diubah menjadi matriks piksel RGB.
- QuadTreeNode = Struktur node pohon untuk merepresentasikan blok gambar.

- ImageCompressor = Class utama untuk mengelola proses kompresi.
- compressedImage = Gambar hasil rekonstruksi dari Quadtree.

Langkah Utama Program

Fungsi compress()

- Menghitung ukuran awal gambar.
- Membuat Quadtree dari gambar.
- Menyusun ulang gambar hasil kompresi.
- Menghitung statistik (ukuran file baru, persentase kompresi, waktu eksekusi, kedalaman pohon, jumlah simpul).

Fungsi buildQuadTree(x, y, width, height)

Divide and Conquer:

- Menghitung nilai error blok dengan metode tertentu.
- Jika error <= threshold atau ukuran blok <= minBlockSize:
    - Blok disimpan sebagai leaf node dengan warna rata-rata.
- Jika tidak memenuhi kondisi di atas:
    - Blok dibagi menjadi 4 sub-blok.
    - buildQuadTree dipanggil rekursif untuk setiap sub-blok.

Fungsi calculateError(x, y, width, height)

Menghitung nilai error untuk blok gambar:

- Variance = Variansi nilai grayscale piksel.
- MAD = Mean Absolute Deviation.
- MaxDiff = Selisih nilai maksimum dan minimum.
- Entropy = Kompleksitas distribusi piksel.
- SSIM = Perbandingan kesamaan blok terhadap blok homogen (1 - SSIM sebagai nilai error).

Fungsi calculateAverageColor(x, y, width, height)

Menghitung warna rata-rata blok untuk diisikan pada gambar hasil kompresi.

Fungsi reconstructImage(node)

Rekonstruksi gambar dari struktur Quadtree:

- Jika node merupakan leaf → Warnai blok dengan averageColor.
- Jika node internal → Rekursif ke setiap child node.

Output Program

Program akan mencetak informasi berikut:

- Waktu eksekusi (dalam milidetik)
- Ukuran gambar sebelum dan sesudah kompresi
- Persentase kompresi
- Kedalaman pohon Quadtree
- Jumlah total simpul pada Quadtree
- Menyimpan gambar hasil kompresi ke alamat output

- (Bonus) Menyimpan GIF visualisasi pembentukan Quadtree

1.3 Karakteristik Program

Dengan pendekatan Divide and Conquer ini:

- Semakin kecil threshold → Semakin banyak pembagian blok → Ukuran file lebih besar, waktu eksekusi lebih lama.
- Semakin besar threshold → Blok cepat menjadi leaf → Ukuran file lebih kecil, tetapi kualitas gambar bisa menurun.
- Metode error yang lebih kompleks (Entropy, SSIM) akan membutuhkan waktu komputasi lebih lama dibandingkan Variance atau MAD.

Program ini murni berbasis rekursi tanpa heuristik percepatan pencarian sehingga untuk gambar besar atau threshold kecil, waktu eksekusi bisa meningkat signifikan.

BAB 2 PROGRAM

Main.java

```
import java.util.Scanner;

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

        // tampilan input
        System.out.print("Input alamat absolut gambar asli: ");
        String inputPath = sc.nextLine();

        System.out.print("Input metode perhitungan error (1=Variance,
2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): ");
        int errorMethod = sc.nextInt();

        System.out.print("Input threshold: ");
        double threshold = sc.nextDouble();

        System.out.print("Input ukuran blok minimum: ");
        int minBlockSize = sc.nextInt();

        System.out.print("Input target persentase kompresi (0.0 jika
nonaktif): ");
        double targetCompression = sc.nextDouble();
        sc.nextLine();

        System.out.print("Input alamat absolut untuk output gambar: ");
        String outputImagePath = sc.nextLine();

        // instansiasi objek kompresi
        ImageCompressor compressor = new ImageCompressor(inputPath,
errorMethod, threshold, minBlockSize, targetCompression,
outputImagePath);
        compressor.compress();

        // tampilan statistik kompresi
```

```
        System.out.println("Waktu eksekusi        : " +
compressor.getExecutionTime() + " ms");
        System.out.println("Ukuran gambar awal    : " +
compressor.getOriginalSize() + " bytes");
        System.out.println("Ukuran gambar kompres : " +
compressor.getCompressedSize() + " bytes");
        System.out.println("Persentase kompresi   : " +
compressor.getCompressionPercentage() + " %");
        System.out.println("Kedalaman pohon       : " +
compressor.getTreeDepth());
        System.out.println("Banyak simpul pohon   : " +
compressor.getTotalNodes());

        sc.close();
    }
}
```

QuadTreeNode.java

```
import java.awt.Color;

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

    // Konstruktor node daun dgn rata2 warna
    public QuadTreeNode(int x, int y, int width, int height, Color
averageColor) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.averageColor = averageColor;
        this.children = null;
    }
```

```
    // Konstruktor node internal dengan anak2
    public QuadTreeNode(int x, int y, int width, int height,
QuadTreeNode[] children) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.children = children;
        this.averageColor = null;
    }

    // Fungsi untuk cek apakah node leaf
    public boolean isLeaf() {
        return (children == null);
    }
}
```

ImageCompressor.java

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

public class ImageCompressor {
    // Parameter input
    private String inputPath;
    private int errorMethod;
    private double threshold;
    private int minBlockSize;
    private String outputImagePath;

    // Variabel untuk nyimpan gambar
    private BufferedImage originalImage;
```

```
    private BufferedImage compressedImage;
    private QuadTreeNode root;

    // Statistik output
    private long executionTime;
    private long originalSize;
    private long compressedSize;
    private int treeDepth;
    private int totalNodes;
    private double compressionPercentage;

    /**
     * Konstruktor ImageCompressor.
     * I.S.: Parameter input terdefinisi
     * F.S.: Objek ImageCompressor terbentuk
     */
    public ImageCompressor(String inputPath, int errorMethod, double
threshold, int minBlockSize, double targetCompression, String
outputImagePath) {
        this.inputPath = inputPath;
        this.errorMethod = errorMethod;
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.outputImagePath = outputImagePath;
    }

    /**
     * Proses utama kompresi gambar dengan Quadtree.
     * I.S.: Gambar yang akan dikompresi tersedia
     * F.S.: Gambar hasil kompresi disimpan beserta statistik output
     */
    public void compress() {
        try {
            // baca gambar dan ukuran
            originalImage = ImageIO.read(new File(inputPath));
            originalSize = new File(inputPath).length();

            long startTime = System.currentTimeMillis();
            // bangun quadtree
```

```
            root = buildQuadTree(0, 0, originalImage.getWidth(),
originalImage.getHeight());

            // image reconstruction from quadtree
            compressedImage = reconstructImage(root,
originalImage.getWidth(), originalImage.getHeight());
            ImageIO.write(compressedImage, "png", new
File(outputImagePath));
            compressedSize = new File(outputImagePath).length();

            // hitung stats kompresi
            compressionPercentage = (1 - ((double)compressedSize /
originalSize)) * 100;
            executionTime = System.currentTimeMillis() - startTime;
            treeDepth = calculateTreeDepth(root);
            totalNodes = countNodes(root);

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

    /**
     * Membangun Quadtree dari blok gambar secara rekursif.
     * I.S.: Gambar original telah dibaca
     * F.S.: Return node Quadtree yang merepresentasikan blok gambar
     */
    private QuadTreeNode buildQuadTree(int x, int y, int width, int
height) {
        double error = calculateError(x, y, width, height);

        // cek kondisi penghentian:
        if (error <= threshold || width <= minBlockSize || height <=
minBlockSize) {
            Color avgColor = calculateAverageColor(x, y, width, height);
            return new QuadTreeNode(x, y, width, height, avgColor);
        } else {
            int halfWidth = width / 2;
            int halfHeight = height / 2;
```

```
            if (halfWidth == 0 || halfHeight == 0) {
                Color avgColor = calculateAverageColor(x, y, width,
height);
                return new QuadTreeNode(x, y, width, height, avgColor);
            }

            QuadTreeNode child1 = buildQuadTree(x, y, halfWidth,
halfHeight);
            QuadTreeNode child2 = buildQuadTree(x + halfWidth, y, width -
halfWidth, halfHeight);
            QuadTreeNode child3 = buildQuadTree(x, y + halfHeight,
halfWidth, height - halfHeight);
            QuadTreeNode child4 = buildQuadTree(x + halfWidth, y +
halfHeight, width - halfWidth, height - halfHeight);

            QuadTreeNode[] children = new QuadTreeNode[]{child1, child2,
child3, child4};
            return new QuadTreeNode(x, y, width, height, children);
        }
    }

    /**
     * Menghitung error pada blok gambar berdasarkan metode yang dipilih.
     * I.S.: Blok gambar terdefinisi oleh (x, y, width, height)
     * F.S.: Mengembalikan nilai error
     *  Metode :
     *  1. Variance (default)
     *  2. Mean Absolute Deviation (MAD)
     *  3. MaxDiff (selisih nilai max min)
     *  4. Entropy
     *  5. Structural Similarity Index (SSIM) - bonus (error = 1 - SSIM)
     */
    private double calculateError(int x, int y, int width, int height) {
        int count = width * height;

        switch (errorMethod) {
            case 1:
                // Variance
```

```
                double sum = 0, sumSq = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        int pixel = (c.getRed() + c.getGreen() +
c.getBlue()) / 3;
                        sum += pixel;
                        sumSq += pixel * pixel;
                    }
                }
                double mean = sum / count;
                return (sumSq / count) - (mean * mean);

            case 2:
                // Mean Absolute Deviation
                double sumVal = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        int pixel = (c.getRed() + c.getGreen() +
c.getBlue()) / 3;
                        sumVal += pixel;
                    }
                }
                double meanVal = sumVal / count;
                            double madSum = 0;
                            for (int i = x; i < x + width; i++) {
                                for (int j = y; j < y + height; j++) {
                                    Color c = new
Color(originalImage.getRGB(i, j));
                                    int pixel = (c.getRed() +
c.getGreen() + c.getBlue()) / 3;
                                    madSum += Math.abs(pixel - meanVal);
                                }
                            }
                            return madSum / count;

            case 3:
                // MaxDiff
```

```
                int minVal = 255, maxVal = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        int pixel = (c.getRed() + c.getGreen() +
c.getBlue()) / 3;
                        if (pixel < minVal) minVal = pixel;
                        if (pixel > maxVal) maxVal = pixel;
                    }
                }
                return maxVal - minVal;

            case 4:
                // Entropy: H = - Σ [p(i) * log2(p(i))]
                int[] histogram = new int[256];
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        int pixel = (c.getRed() + c.getGreen() +
c.getBlue()) / 3;
                        histogram[pixel]++;
                    }
                }
                double entropy = 0;
                for (int i = 0; i < 256; i++) {
                    if (histogram[i] > 0) {
                        double p = (double) histogram[i] / count;
                        entropy += p * (Math.log(p) / Math.log(2));  //
log base 2
                    }
                }
                return -entropy;

            case 5:
                // Structural Similarity Index (SSIM)
                double sumR = 0, sumG = 0, sumB = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
```

```
                        sumR += c.getRed();
                        sumG += c.getGreen();
                        sumB += c.getBlue();
                    }
                }
                double avgR = sumR / count;
                double avgG = sumG / count;
                double avgB = sumB / count;

                double varR = 0, varG = 0, varB = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        varR += Math.pow(c.getRed() - avgR, 2);
                        varG += Math.pow(c.getGreen() - avgG, 2);
                        varB += Math.pow(c.getBlue() - avgB, 2);
                    }
                }
                varR /= count;
                varG /= count;
                varB /= count;

                double C2 = Math.pow(0.03 * 255, 2);
                double ssimR = C2 / (varR + C2);
                double ssimG = C2 / (varG + C2);
                double ssimB = C2 / (varB + C2);
                double avgSSIM = (ssimR + ssimG + ssimB) / 3.0;
                return 1.0 - avgSSIM;

            default:
                double defSum = 0, defSumSq = 0;
                for (int i = x; i < x + width; i++) {
                    for (int j = y; j < y + height; j++) {
                        Color c = new Color(originalImage.getRGB(i, j));
                        int pixel = (c.getRed() + c.getGreen() +
c.getBlue()) / 3;
                        defSum += pixel;
                        defSumSq += pixel * pixel;
                    }
```

```
                }
                double defMean = defSum / count;
                return (defSumSq / count) - (defMean * defMean);
            }
        }

        /**
         * Menghitung rata-rata warna pada blok.
         * I.S.: Blok gambar terdefinisi oleh (x, y, width, height)
         * F.S.: Mengembalikan nilai Color rata-rata untuk blok
         */
        private Color calculateAverageColor(int x, int y, int width, int
height) {
            long sumR = 0, sumG = 0, sumB = 0;
            int count = width * height;

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

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

        /**
         * Rekonstruksi gambar terkompresi dari struktur Quadtree.
         * I.S.: Quadtree telah terbentuk
         * F.S.: Mengembalikan BufferedImage hasil rekonstruksi
         */
        private BufferedImage reconstructImage(QuadTreeNode node, int width,
int height) {
            BufferedImage image = new BufferedImage(width, height,
```

```
BufferedImage.TYPE_INT_RGB);
        reconstructHelper(node, image);
        return image;
    }

    // Fungsi rekursif untuk mewarnai gambar
    private void reconstructHelper(QuadTreeNode node, BufferedImage
image) {
        if (node.isLeaf()) {
            for (int i = node.x; i < node.x + node.width; i++) {
                for (int j = node.y; j < node.y + node.height; j++) {
                    image.setRGB(i, j, node.averageColor.getRGB());
                }
            }
        } else {
            for (QuadTreeNode child : node.children) {
                reconstructHelper(child, image);
            }
        }
    }

    /**
     * Menghitung kedalaman pohon Quadtree.
     * I.S.: Quadtree telah terbentuk
     * F.S.: Return maxDepth pohon
     */
    private int calculateTreeDepth(QuadTreeNode node) {
        if (node.isLeaf()) {
            return 1;
        } else {
            int maxDepth = 0;
            for (QuadTreeNode child : node.children) {
                int childDepth = calculateTreeDepth(child);
                if (childDepth > maxDepth) {
                    maxDepth = childDepth;
                }
            }
            return maxDepth + 1;
        }
```

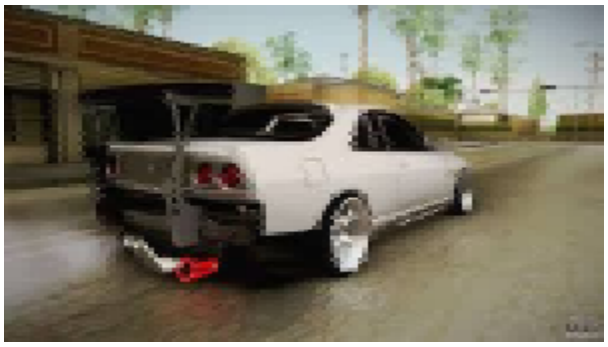
```
    }

    /**
     * Menghitung jumlah total simpul pada Quadtree.
     * I.S.: Quadtree telah terbentuk
     * F.S.: Return jumlah node pada pohon
     */
    private int countNodes(QuadTreeNode node) {
        if (node.isLeaf()) {
            return 1;
        } else {
            int count = 1;
            for (QuadTreeNode child : node.children) {
                count += countNodes(child);
            }
            return count;
        }
    }

    // Getter untuk output stats
    public long getExecutionTime() {
        return executionTime;
    }

    public long getOriginalSize() {
        return originalSize;
    }

    public long getCompressedSize() {
        return compressedSize;
    }

    public double getCompressionPercentage() {
        return compressionPercentage;
    }

    public int getTreeDepth() {
        return treeDepth;
    }
```

```
    public int getTotalNodes() {
        return totalNodes;
    }
}
```

BAB 3 EKSPERIMEN

<table>
<tr><th colspan="2">Input</th></tr>
<tr><td>1.</td><td>

```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input threshold: 5
Input ukuran blok minimum: 2
Input target persentase kompresi (0.0 jika nonaktif): 0
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\compressed_images1.jpg
Waktu eksekusi        : 92 ms
Ukuran gambar awal    : 8352 bytes
Ukuran gambar kompres : 36499 bytes
Kedalaman pohon       : 8
Banyak simpul pohon   : 13697
PS C:\Users\Mykomp\Tucil2_13523150\bin> cd ../src
PS C:\Users\Mykomp\Tucil2_13523150\src> pause
Press Enter to continue...:
```

</td></tr>
<tr><th colspan="2">Output</th></tr>
<tr><td></td><td>  </td></tr>
<tr><th colspan="2">Input</th></tr>
<tr><td>2.</td><td>

```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 1
Input threshold: 500
Input ukuran blok minimum: 2
Input target persentase kompresi (0.0 jika nonaktif): 0
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Tucil2_13523150\test\compressed_images2.jpg
Waktu eksekusi        : 112 ms
Ukuran gambar awal    : 8352 bytes
Ukuran gambar kompres : 10433 bytes
Persentase kompresi   : -24.916187739463602 %
Kedalaman pohon       : 8
Banyak simpul pohon   : 2609
Press any key to continue . . .
```

</td></tr>
<tr><th colspan="2">Output</th></tr>
</table>

Input

3.

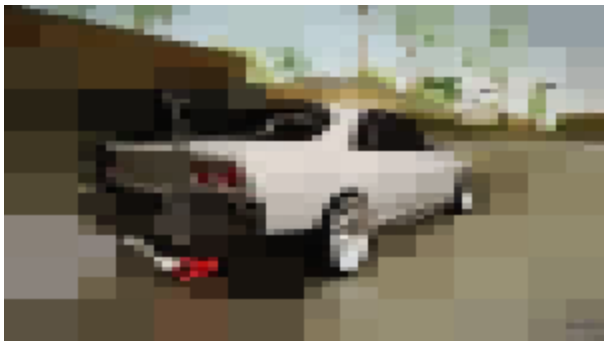
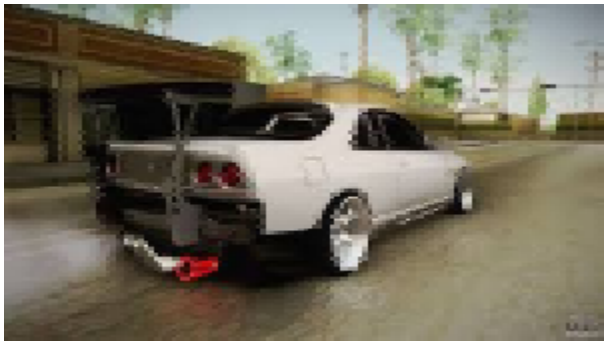
```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 2
Input threshold: 50
Input ukuran blok minimum: 4
Input target persentase kompresi (0.0 jika nonaktif): 0
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\compressed_images3.jpg
Waktu eksekusi        : 70 ms
Ukuran gambar awal    : 8352 bytes
Ukuran gambar kompres : 1503 bytes
Persentase kompresi   : 82.00431034482759 %
Kedalaman pohon       : 7
Banyak simpul pohon   : 41
Press any key to continue . . .
```

Output

Input

4.

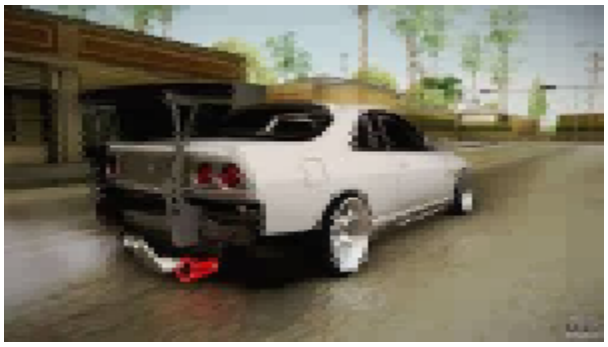
```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 3
Input threshold: 100
Input ukuran blok minimum: 4
Input target persentase kompresi (0.0 jika nonaktif): 0
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Tucil2_13523150\test\compressed_images4.jpg
Waktu eksekusi        : 82 ms
Ukuran gambar awal    : 8352 bytes
Ukuran gambar kompres : 6960 bytes
Persentase kompresi   : 16.666666666666664 %
Kedalaman pohon       : 7
Banyak simpul pohon   : 1481
```

Output	
	
Input	
5.	Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 4 Input threshold: 0.8 Input ukuran blok minimum: 2 Input target persentase kompresi (0.0 jika nonaktif): 0 Input alamat absolut untuk output gambar: C:\Users\Mykomp\Tucil2_13523150\test\compressed_images5.jpg Waktu eksekusi : 110 ms Ukuran gambar awal : 8352 bytes Ukuran gambar kompres : 38662 bytes Persentase kompresi : -362.90708812260533 % Kedalaman pohon : 8 Banyak simpul pohon : 15637
Output	
	
Input	

6.

```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 5
Input threshold: 0.2
Input ukuran blok minimum: 2
Input target persentase kompresi (0.0 jika nonaktif): 0
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Tucil2_13523150\test\compressed_images6.jpg
Waktu eksekusi         : 115 ms
Ukuran gambar awal     : 8352 bytes
Ukuran gambar kompres : 33198 bytes
Persentase kompresi    : -297.4856321839081 %
Kedalaman pohon        : 8
Banyak simpul pohon    : 11825
```

Output

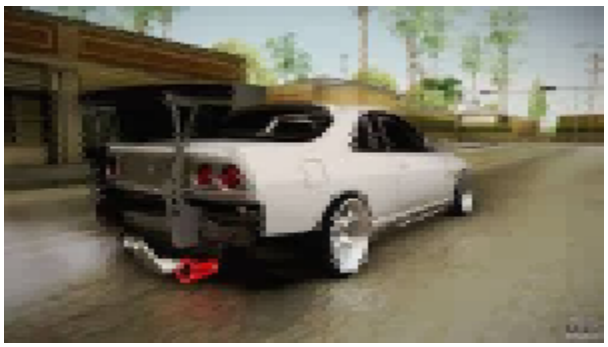


Input

7.

```
Input alamat absolut gambar asli: C:\Users\Mykomp\Documents\Photos, Videos\Wallpaper and Profiles\images.jpg
Input metode perhitungan error (1=Variance, 2=MAD, 3=MaxDiff, 4=Entropy, 5=SSIM): 1
Input threshold: 10
Input ukuran blok minimum: 2
Input target persentase kompresi (0.0 jika nonaktif): 0.5
Input alamat absolut untuk output gambar: C:\Users\Mykomp\Tucil2_13523150\test\compressed_images7.jpg
Waktu eksekusi         : 91 ms
Ukuran gambar awal     : 8352 bytes
Ukuran gambar kompres : 34383 bytes
Persentase kompresi    : -311.6738505747127 %
Kedalaman pohon        : 8
Banyak simpul pohon    : 12477
```

Output



LAMPIRAN

Link Repository

https://github.com/BenedictusNelson/Tucil2_13523150.git

checklist

No	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 (Variance, MAD, MaxDiff, Entropy)	✓	
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	✓	