

LAPORAN TUGAS KECIL II

IF2211 - STRATEGI ALGORITMA



Disusun oleh :

Barru Adi Utomo – 13523101

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I SPESIFIKASI TUGAS.....	3
BAB 2 LANDASAN TEORI.....	5
1. Error Measurement Methods (Metode Pengukuran Error).....	5
2. Threshold (Ambang Batas).....	7
3. Minimum Block Size (Ukuran Blok Minimum).....	7
4. Compression Percentage (Persentase Kompresi) [BONUS].....	7
BAB 3 IMPLEMENTASI.....	8
A. Main program.....	8
B. ErrorMethod.....	11
a. Variance.....	11
b. MAD.....	13
c. MPD.....	14
d. Entropy.....	15
e. SSIM.....	16
C. Input dan Output.....	17
D. Utils.....	24
BAB 4 PENGUJIAN.....	26
Pengujian 1.....	26
Pengujian 2.....	26
Pengujian 3.....	27
Pengujian 4.....	28
Pengujian 5.....	28
Pengujian 6.....	29
Pengujian 7.....	30
LAMPIRAN.....	32

BAB I

SPESIFIKASI TUGAS

- Buatlah program sederhana dalam bahasa **C/C#/C++/Java** (CLI) yang mengimplementasikan **algoritma *divide and conquer*** untuk melakukan kompresi gambar berbasis *quadtree* yang mengimplementasikan **seluruh parameter** yang telah disebutkan sebagai *user input*.
- Alur program:
 1. [INPUT] **alamat absolut** gambar yang akan dikompresi.
 2. [INPUT] metode perhitungan error (gunakan penomoran sebagai *input*).
 3. [INPUT] ambang batas (pastikan *range* nilai sesuai dengan metode yang dipilih).
 4. [INPUT] ukuran blok minimum.
 5. [INPUT] Target persentase kompresi (*floating number*, 1.0 = 100%), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
 6. [INPUT] **alamat absolut** gambar hasil kompresi.
 7. [INPUT] **alamat absolut** gif (bonus).
 8. [OUTPUT] waktu eksekusi.
 9. [OUTPUT] ukuran gambar sebelum.
 10. [OUTPUT] ukuran gambar setelah.
 11. [OUTPUT] persentase kompresi.
 12. [OUTPUT] kedalaman pohon.
 13. [OUTPUT] banyak simpul pada pohon.
 14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
 15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).
- Berkas laporan yang dikumpulkan adalah laporan dalam bentuk PDF yang setidaknya berisi:
 1. Algoritma ***divide and conquer*** yang digunakan, jelaskan langkah-langkahnya, **bukan hanya notasi pseudocode dan BUKAN IMPLEMENTASINYA melainkan algoritmanya**.
 2. *Source* program dalam bahasa pemrograman yang dipilih (pastikan bahwa program telah dapat dijalankan).
 3. Tangkapan layar yang memperlihatkan *input* dan *output* (minimal sebanyak 7 buah contoh).
 4. Hasil analisis percobaan algoritma ***divide and conquer*** dalam kompresi gambar dengan metode Quadtree. Analisis dilakukan dalam bentuk paragraf/poin dan

minimal memuat mengenai analisis kompleksitas algoritma program yang telah dikembangkan.

5. Penjelasan mengenai implementasi bonus jika mengerjakan.
6. Pranala ke *repository* yang berisi kode program.

- **BONUS:**

Pastikan sudah mengerjakan spesifikasi wajib sebelum mengerjakan bonus:

1. Memungkinkan pengguna menentukan **target persentase kompresi** berupa nilai floating point (contoh: 1.0 = 100%). Jika pengguna memberikan nilai 0, mode ini akan dinonaktifkan. Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai threshold secara otomatis untuk mencapai target persentase kompresi yang ditentukan. Penyesuaian threshold ini dilakukan secara dinamis, memberikan fleksibilitas dalam proses kompresi untuk memenuhi target efisiensi sambil mempertahankan kualitas gambar.
2. Implementasikan **Structural Similarity Index (SSIM)** sebagai metode pengukuran error. Hitung SSIM untuk setiap kanal warna R, G, dan B, lalu gabungkan hasilnya menjadi SSIM total menggunakan bobot masing-masing kanal. Informasi mengenai formula perhitungan dan parameter SSIM dapat ditemukan di *Tabel 1*.
3. Membuat **GIF** proses kompresi seperti yang tertera pada **Gambar 2**. Visualisasi proses Pembentukan Quadtree dalam Kompresi Gambar dengan format **GIF**.

BAB 2

LANDASAN TEORI

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

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

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika error dalam blok melebihi ambang batas (*threshold*), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Tabel 1. Metode Pengukuran Error

Metode	Formula
<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c
<u>Entropy</u>	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok

	<p>blok</p> $P_c(i) = \text{Probabilitas piksel dengan nilai } i \text{ dalam satu blok untuk tiap kanal warna } c \text{ (R, G, B)}$
<p>[Bonus]</p> <p>Structural Similarity Index (SSIM)</p> <p>(Referensi tambahan)</p>	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	<p>Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.</p>

2. Threshold (Ambang Batas)

Threshold adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.

3. Minimum Block Size (Ukuran Blok Minimum)

Minimum block size (luas piksel) adalah ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Jika ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah ukuran minimum yang telah dikonfigurasi, maka blok tersebut tidak akan dibagi lebih lanjut, meskipun error masih di atas threshold.

4. Compression Percentage (Persentase Kompresi) [BONUS]

Persentase kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

$$\text{Persentase Kompresi} = \left(1 - \frac{\text{Ukuran Gambar Terkompresi}}{\text{Ukuran Gambar Asli}}\right) \times 100\%$$

BAB 3

IMPLEMENTASI

Penjelasan:

Algoritma Quadtree Compression dengan pendekatan divide and conquer merupakan metode kompresi citra yang bekerja dengan membagi citra menjadi empat bagian secara rekursif berdasarkan tingkat keseragaman warna dalam suatu blok. Proses dimulai dengan memeriksa apakah seluruh piksel dalam suatu area memiliki nilai warna yang cukup mirip atau homogen, berdasarkan ukuran error tertentu seperti varians, MAD (Mean Absolute Deviation), SSIM, atau entropi. Jika blok tersebut dianggap homogen, maka blok tidak akan dibagi lagi dan disimpan sebagai simpul daun (leaf node) dalam struktur pohon. Namun jika blok tersebut tidak homogen, maka area tersebut akan dibagi menjadi empat kuadran: kiri atas, kanan atas, kiri bawah, dan kanan bawah. Proses ini diulang secara rekursif untuk setiap kuadran, sehingga membentuk struktur pohon (quadtree) di mana setiap simpul dapat mewakili bagian citra yang telah disederhanakan. Pendekatan ini efisien untuk citra yang memiliki area besar dengan warna seragam karena mampu mengurangi jumlah data yang perlu disimpan tanpa mengorbankan detail visual yang penting.

A. Main program

Main
<pre>// IMPORT ALL FILES import java.awt.image.BufferedImage; import java.io.IOException; import javax.imageio.ImageIO; import java.io.File; import Business.*; import Utils.*; public class Main { public static void main(String[] args) { Utils.clearScreen(); try { // ----- INITILIZE ----- ImageTree tree; BufferedImage outputImage; int treeDepth; int totalNode; // ----- INPUT ----- CLIio cli = new CLIio();</pre>


```

cli.cliInput();
boolean targetKompresiStatus = cli.getTargetKompresiStatus();
float minThreshold = 0;
float maxThreshold = cli.getMaxThreshold();
float midThreshold = (minThreshold + maxThreshold) / 2;

long startTime = System.currentTimeMillis();           // Start
Timer

// ----- PROSES -----
BufferedImage image = cli.getImage();

if (targetKompresiStatus) {
    while(true) {
        tree = new ImageTree(
            image,
            midThreshold,
            cli.getInputBlockSize(),
            cli.getInputMethod()
        );
        outputImage = new BufferedImage(
            image.getWidth(),
            image.getHeight(),
            BufferedImage.TYPE_INT_RGB
        );
        tree.reconstructImage(outputImage);
        String fileExtension = cli.getFileExtension();
        ImageIO.write(outputImage, fileExtension, new
File(cli.getOutputPath()));
        double output_file_size_before = Utils.getFileSizeKb(new
File(cli.getInputPath()));
        double output_file_size_after = Utils.getFileSizeKb(new
File(cli.getOutputPath()));
        double output_compression_percentage = (1 - (double)
output_file_size_after / output_file_size_before) * 100;

        if (Math.abs(output_compression_percentage -
cli.getInputTargetCompression()) < 0.01) {
            treeDepth = tree.calculateDepth();
            totalNode = tree.getTotalNode();
            break;
        } else if (Math.abs(maxThreshold - minThreshold) < 0.01) {
            System.out.println("ERR: Target compression not

```

```

    achieved");

        treeDepth = tree.calculateDepth();
        totalNode = tree.getTotalNode();
        break;
    }

    if (output_compression_percentage <
cli.getInputTargetCompression()) {
        maxThreshold = midThreshold;
        midThreshold = (minThreshold + maxThreshold) / 2;
    } else if (output_compression_percentage >
cli.getInputTargetCompression()) {
        minThreshold = midThreshold;
        midThreshold = (minThreshold + maxThreshold) / 2;
    }

    File outputFile = new File(cli.getOutputPath());
    if (outputFile.exists()) {
        outputFile.delete();
    }
}
} else {
    tree = new ImageTree(
        image,
        cli.getInputThreshold(),
        cli.getInputBlockSize(),
        cli.getInputMethod()
    );

    outputImage = new BufferedImage(
        image.getWidth(),
        image.getHeight(),
        BufferedImage.TYPE_INT_RGB
    );
    tree.reconstructImage(outputImage);
    treeDepth = tree.calculateDepth();
    totalNode = tree.getTotalNode();

    String fileExtension = cli.getFileExtension();
    ImageIO.write(outputImage, fileExtension, new
File(cli.getOutputPath()));
}
}

```

```

        long endTime = System.currentTimeMillis();                // End
Timer

        // ----- OUTPUT -----
        cli.cliOutput(
            endTime - startTime,
            treeDepth,
            totalNode
        );

    } catch (IOException e) {
        System.out.println("ERR: " + e);
        return;
    }
}
}

```

B. ErrorMethod

a. Variance

Variance Method
<pre> package ErrorMethods; import java.awt.Color; import java.awt.image.BufferedImage; public class VarianceErrorMethod implements _ErrorMethod { @Override public double calculateErr(BufferedImage image, int x, int y, int width, int height) { //-----Initialisasi----- int total_pixels = width * height; long sum_R = 0; double var_R = 0; long sum_G = 0; double var_G = 0; long sum_B = 0; double var_B = 0; //-----jumlah pixel dalmm satu blok----- for (int i = x; i <= x + width; i++) { for (int j = y; j <= y + height; j++) { if (i < 0 i >= image.getWidth() j < 0 j >= </pre>

```

image.getHeight()) {
    continue;
}
Color color = new Color(image.getRGB(i, j));
sum_R += color.getRed();
sum_G += color.getGreen();
sum_B += color.getBlue();
}
}

//-----Rata" Pixel dalam satu block-----
double mean_R = sum_R / (double) total_pixels;
double mean_G = sum_G / (double) total_pixels;
double mean_B = sum_B / (double) total_pixels;

// -----Variansi setiap kanal warna-----
for (int i = x; i <= x + width; i++) {
    for (int j = y; j <= y + height; j++) {
        if (i < 0 || i >= image.getWidth() || j < 0 || j >=
image.getHeight()) {
            continue;
        }
        Color color = new Color(image.getRGB(i, j));
        var_R += Math.pow(color.getRed() - mean_R, 2);
        var_G += Math.pow(color.getGreen() - mean_G, 2);
        var_B += Math.pow(color.getBlue() - mean_B, 2);
    }
}

var_R /= total_pixels;
var_G /= total_pixels;
var_B /= total_pixels;

//-----Variansi total-----
double total_variance = (var_R + var_G + var_B) / 3.0;

return total_variance;
}
}

```

b. MAD

MAD method

```
package ErrorMethods;

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

public class MADErrorMethod implements _ErrorMethod{
    @Override
    public double calculateErr(BufferedImage image, int x, int y, int width,
int height) {
        //-----Inisialisasi-----
        int total_pixels = width * height;
        long sum_R = 0; double mad_R = 0;
        long sum_G = 0; double mad_G = 0;
        long sum_B = 0; double mad_B = 0;

        //-----jumlah pixel dalam satu blok-----
        for (int i = x; i <= x + width; i++) {
            for (int j = y; j <= y + height; j++) {
                Color color = new Color(image.getRGB(i, j));
                sum_R += color.getRed();
                sum_G += color.getGreen();
                sum_B += color.getBlue();
            }
        }

        //-----Rata" Pixel dalam satu block-----
        double mean_R = sum_R / (double) total_pixels;
        double mean_G = sum_G / (double) total_pixels;
        double mean_B = sum_B / (double) total_pixels;

        // -----MAD setiap kanal warna-----
        for (int i = x; i <= x + width; i++) {
            for (int j = y; j <= y + height; j++) {
                Color color = new Color(image.getRGB(i, j));
                mad_R += Math.abs(color.getRed() - mean_R);
                mad_G += Math.abs(color.getGreen() - mean_G);
                mad_B += Math.abs(color.getBlue() - mean_B);
            }
        }
    }
}
```

```

        mad_R /= total_pixels;
        mad_G /= total_pixels;
        mad_B /= total_pixels;

        //-----MAD total-----
        double total_mad = (mad_R + mad_G + mad_B) / 3.0;

        return total_mad;
    }
}

```

c. MPD

Main
<pre> package ErrorMethods; import java.awt.Color; import java.awt.image.BufferedImage; public class MPDErrorMethod implements _ErrorMethod { @Override public double calculateErr(BufferedImage image, int x, int y, int width, int height) { //-----Initialisasi----- int max_R = 0; int min_R = 256; int max_G = 0; int min_G = 256; int max_B = 0; int min_B = 256; //-----Milai Max dan Min----- for (int i = x; i <= x + width; i++) { for (int j = y; j <= y + height; j++){ Color color = new Color(image.getRGB(i, j)); // cari max max_R = Math.max(max_R, color.getRed()); max_G = Math.max(max_G, color.getGreen()); max_B = Math.max(max_B, color.getBlue()); // cari min min_R = Math.min(min_R, color.getRed()); min_G = Math.min(min_G, color.getGreen()); min_B = Math.min(min_B, color.getBlue()); } } } } </pre>

```

    }
}

// -----Hitung max diff-----
int delta_R = max_R - min_R;
int delta_G = max_G - min_G;
int delta_B = max_B - min_B;

double delta_total = ((double) (delta_R + delta_G + delta_B)) / 3.0;

return delta_total;
}
}

```

d. Entropy

Entropi Metode

```

package ErrorMethods;

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

public class EntropyErrorMethod implements _ErrorMethod {
    @Override
    public double calculateErr(BufferedImage image, int x, int y, int width,
int height) {
        //-----Inisialisasi-----
        long total_pixels = width * height;
        int[] probability_R = new int[256]; double entropy_R = 0;
        int[] probability_G = new int[256]; double entropy_G = 0;
        int[] probability_B = new int[256]; double entropy_B = 0;

        //-----Hitung probabilitas-----
        for (int i = x; i <= x + width; i++) {
            for (int j = y; j <= y + height; j++) {
                Color color = new Color(image.getRGB(i, j));
                probability_R[color.getRed()]++;
                probability_G[color.getGreen()]++;
                probability_B[color.getBlue()]++;
            }
        }
    }
}

```

```

//-----Hitung Entropi-----
for (int i = 0; i < 256; i++) {
    if (probability_R[i] > 0) {
        double p = (double) probability_R[i] / total_pixels;
        entropy_R -= p * Math.log(p) / Math.log(2);
    }
    if (probability_G[i] > 0) {
        double p = (double) probability_G[i] / total_pixels;
        entropy_G -= p * Math.log(p) / Math.log(2);
    }
    if (probability_B[i] > 0) {
        double p = (double) probability_B[i] / total_pixels;
        entropy_B -= p * Math.log(p) / Math.log(2);
    }
}

//-----Hitung Entropi Total-----
double total_entropi = (entropy_R + entropy_G + entropy_B) / 3.0;

return total_entropi;
}
}

```

e. SSIM

Main
<pre> package ErrorMethods; import java.awt.Color; import java.awt.image.BufferedImage; public class SSIMErrorMethod implements _ErrorMethod { @Override public double calculateErr(BufferedImage image, int x, int y, int width, int height) { if (width < 2 height < 2) return 1.0; double meanR = 0, meanG = 0, meanB = 0; int totalPixels = width * height; // First pass: compute means for (int i = x; i < x + width; i++) { </pre>


```

        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            meanR += c.getRed();
            meanG += c.getGreen();
            meanB += c.getBlue();
        }
    }
    meanR /= totalPixels;
    meanG /= totalPixels;
    meanB /= totalPixels;

    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(image.getRGB(i, j));
            varR += Math.pow(c.getRed() - meanR, 2);
            varG += Math.pow(c.getGreen() - meanG, 2);
            varB += Math.pow(c.getBlue() - meanB, 2);
        }
    }

    varR /= totalPixels;
    varG /= totalPixels;
    varB /= totalPixels;

    double C1 = 6.5025;

    double ssimR = (2 * meanR * meanR + C1) / (meanR * meanR + varR + C1);
    double ssimG = (2 * meanG * meanG + C1) / (meanG * meanG + varG + C1);
    double ssimB = (2 * meanB * meanB + C1) / (meanB * meanB + varB + C1);

    // average SSIM
    double avgSSIM = (ssimR + ssimG + ssimB) / 3.0;

    return avgSSIM;
}
}

```

C. Input dan Output

Main

```

package Utils;

import java.util.Locale;
import java.util.Scanner;
import javax.imageio.ImageIO;
import java.io.File;
import java.io.IOException;
import java.text.NumberFormat;
import java.text.ParseException;
import java.awt.image.BufferedImage;

public class CLiio {

    private BufferedImage    image;

    //-----INPUT-----
    private String          input;
    private Number          num;
    private String          input_path;
    private Integer         input_method;
    private Float           input_threshold;
    private Integer         input_block_size;
    private Float           input_target_compression;

    //-----OUTPUT-----
    private String          output_path;
    private double          output_file_size_before;
    private double          output_file_size_after;
    private double          output_compression_percentage;

    //-----help (me)-----
    private final boolean   INPUT_VALIDATION        = true;
    private final boolean   INPUT_PATH_ABSOLUTE     = false; // for test n
debug
    private final boolean   OUTPUT_PATH_ABSOLUTE    = false; // for test n
debug
    private final boolean   BONUS_TARGET_KOMPRESI   = true; // [ BONUS ] target
kompresi
    private boolean        TARGET_KOMPRESI         = false;

    public void cliInput() throws IOException {

```

```

Scanner scanner = new Scanner(System.in);

// Float Handling :]
Locale userLocale = Locale.getDefault();
NumberFormat format = NumberFormat.getInstance(userLocale);

//-----ALAMAT ABSOLUT-----
System.out.print("Path gambar: ");
String input_filename = scanner.nextLine();

while (!fileExist(input_filename)) {                                // file
tidak valid
    System.out.println("ERR: File tidak ditemukan");
    System.out.print("Path gambar: ");
    input_filename = scanner.nextLine();
}
// read image
if (INPUT_PATH_ABSOLUTE) {
    input_path = input_filename;
} else {
    input_path = "test/input/" + input_filename;
}
image = ImageIO.read(new File(input_path));

//-----METODE PERHITUNGAN-----
// 1. Variance 2. MAD 3. MPD 4. Entropy
while (INPUT_VALIDATION) {
    System.out.print("Metode perhitungan: ");
    input = scanner.nextLine();
    try {
        input_method = Integer.parseInt(input);
        if (input_method < 1 || input_method > 5) {                //
metode tidak valid
            System.out.println("ERR: Metode tidak valid");
            continue;
        }
        break;

    } catch (NumberFormatException e) {                            // tipe
data tidak betul
        System.out.println("Input tidak valid!");
        continue;
    }
}

```

```

    }
}

//-----TRESHOLD-----
while (INPUT_VALIDATION) {
    System.out.print("Threshold: ");
    input = scanner.nextLine();
    try {
        num = format.parse(input);
        input_threshold = num.floatValue();
        // if (input_threshold < 0 || input_threshold > 1) {           //
threshold tidak valid
        //     System.out.println("ERR: Threshold tidak valid");
        //     continue;
        // }
        break;
    } catch (ParseException e) {                                     // tipe data
tidak betul
        System.out.println("Input tidak valid!");
        continue;
    }
}

//-----UKURAN BLOK MINIMUM-----
while (INPUT_VALIDATION) {
    System.out.print("Ukuran blok minimum: ");
    input = scanner.nextLine();
    try {
        input_block_size = Integer.parseInt(input);
        if (input_block_size < 1) {                                   //
ukuran blok tidak valid
        System.out.println("ERR: Ukuran blok tidak valid");
        continue;
    }
    break;
    } catch (NumberFormatException e) {                             // tipe
data tidak betul
        System.out.println("Input tidak valid!");
        continue;
    }
}

```

```

    }
}

//-----[BONUS] TARGET KOMPRESI-----
if (BONUS_TARGET_KOMPRESI)
{
    while (INPUT_VALIDATION) {
        System.out.print("Target presentase kompresi: ");
        input = scanner.nextLine();
        try {
            num = format.parse(input);
            input_target_compression = num.floatValue();
            if (input_target_compression < 0 ||
input_target_compression > 1) { // target kompresi tidak valid
                System.out.println("ERR: Target presentase kompresi
tidak valid");
                    continue;
                }
                if (input_target_compression == 0) {
                    TARGET_KOMPRESI = false;
                } else {
                    TARGET_KOMPRESI = true;
                }
                break;
            } catch (ParseException e) { // tipe
data tidak betul
                System.out.println("Input tidak valid!");
                continue;
            }
        }
    }
}

//-----ALAMAT OUTPUT-----
while (INPUT_VALIDATION) {
    System.out.print("Path output: ");
    input = scanner.nextLine();
    if (OUTPUT_PATH_ABSOLUTE) {
        output_path = input;
        if (!dirExists(output_path)) { //
direktori tidak valid

```

```

        System.out.println("ERR: Direktori tidak ditemukan");
        continue;
    }
} else {
    output_path = "test/output/" + input;
}
break;
}

scanner.close();
return;
}

public void cliOutput(long executionTime, int treeDepth, int totalNode) {
    // WAKTU EKSEKUSI
    System.out.println("Waktu eksekusi: " + executionTime + " ms");

    // UKURAN GAMBAR SEBELUM
    output_file_size_before = Utils.getFileSizeKb(new
File(getInputPath()));
    System.out.println("Ukuran gambar sebelum: " +
output_file_size_before);

    // UKURAN GAMBAR SESUDAH
    output_file_size_after = Utils.getFileSizeKb(new
File(getOutputPath()));
    System.out.println("Ukuran gambar sesudah: " + output_file_size_after);

    // PRESENTASE KOMPRESI
    output_compression_percentage =
        (1 - (double) output_file_size_after / output_file_size_before) *
100;
    System.out.println("Presentase kompresi: " +
output_compression_percentage);

    // KEDALAMAN POHON
    System.out.println("Kedalaman pohon: " + treeDepth);

    // BANYAK SIMPUL
    System.out.println("Banyak simpul: " + totalNode);

    // [ BONUS ] GIF KOMPRESI

```

```

        return;
    }

    //----- HELPER -----
    public boolean fileExist(String filename) {
        if (INPUT_PATH_ABSOLUTE) {
            File file = new File(filename);
            return file.isFile();
        }
        return new File("test/input/" + filename).isFile();
    };

    public String getFileExtension() {
        int dotIndex = input_path.lastIndexOf(".");
        if (dotIndex != -1 && dotIndex < input_path.length() - 1) {
            return input_path.substring(dotIndex + 1).toLowerCase();
        }
        return null;
    }

    public boolean dirExists(String path) {
        File file = new File(path);
        return file.exists() && file.isDirectory();
    }

    public float getMaxThreshold() {
        if (input_method == 1) { // Variance
            return 16256.25f;
        } else if (input_method == 2) { // MAD
            return 127.5f;
        } else if (input_method == 3) { // MPD
            return 255.0f;
        } else if (input_method == 4) { // Entropy
            return 8.0f;
        } else if (input_method == 5) { // SSIM
            return 1.0f;
        }
        return 0;
    }

    //----- GETTER -----

```

```

public BufferedImage getImage() {
    return image;
}

public String getInputPath() {
    return input_path;
}

public Integer getInputMethod() {
    return input_method;
}

public Float getInputThreshold() {
    return input_threshold;
}

public Integer getInputBlockSize() {
    return input_block_size;
}

public Float getInputTargetCompression() {
    return input_target_compression;
}

public String getOutputPath() {
    return output_path;
}

public boolean getTargetKompresiStatus() {
    return TARGET_KOMPRESI;
}
}

```

D. Utils

Main
<pre> package Utils; import java.io.File; public class Utils { public static void clearScreen() { </pre>


```
        System.out.print("\033[H\033[2J");
        System.out.flush();
    }


    public static double getFileSizeKb(File file) {
        return (double) file.length() / 1024;
    }

    public static void printOutput(String thing, String... strings) {
        System.out.print(thing);
        for (String string : strings) {
            System.out.print(string);
        }
        System.out.println();
    }
}
```

BAB 4

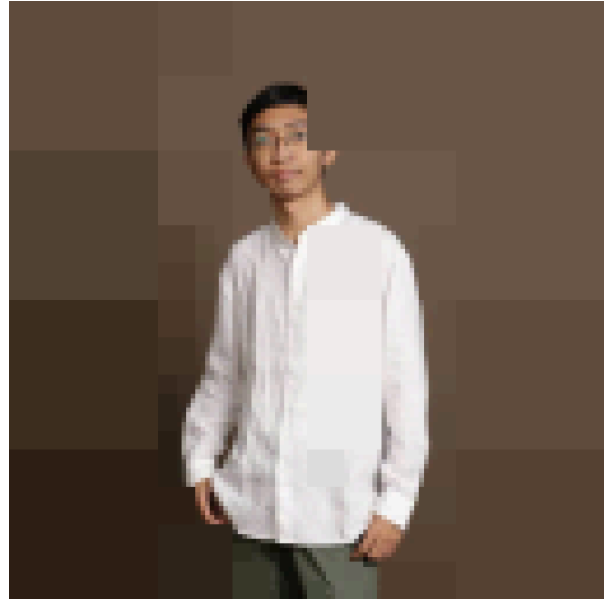
PENGUJIAN

Pengujian 1

Gambar sebelum	Gambar setelah
	
Hasil Input dan Output	
<pre>Path gambar: /Users/barruadi/Documents/random2/test_input.png Metode perhitungan: 1 Threshold: 150 Ukuran blok minimum: 240 Target presentase kompresi: 0 Path output: /Users/barruadi/Documents/random2/test_output1.png Waktu eksekusi: 420 ms Ukuran gambar sebelum: 254.3115234375 Ukuran gambar sesudah: 40.1953125 Presentase kompresi: 84.1944588445366 Kedalaman pohon: 6 Banyak simpul: 1313</pre>	

Pengujian 2

Gambar sebelum	Gambar setelah
----------------	----------------



Hasil Input dan Output

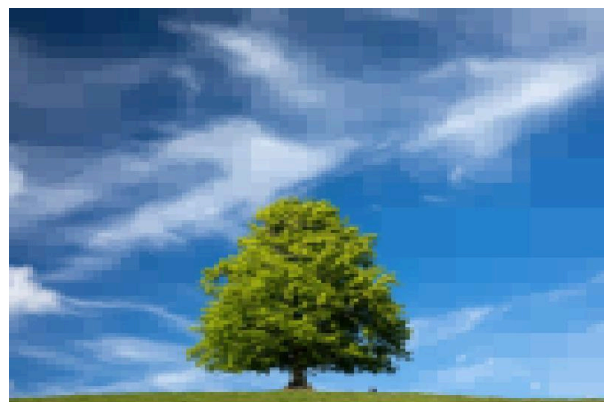
```
Path gambar: /Users/barruadi/Documents/random2/test_input.png
Metode perhitungan: 2
Threshold: 10
Ukuran blok minimum: 120
Target presentase kompresi: 0
Path output: /Users/barruadi/Documents/random2/test_output12.png
Waktu eksekusi: 421 ms
Ukuran gambar sebelum: 254.3115234375
Ukuran gambar sesudah: 27.8359375
Presentase kompresi: 89.05439394812127
Kedalaman pohon: 7
Banyak simpul: 4505
```

Pengujian 3

Gambar sebelum



Gambar setelah





Hasil Input dan Output

```

Path gambar: /Users/barruadi/Documents/random2/test_input3.jpg
Metode perhitungan: 3
Threshold: 30
Ukuran blok minimum: 64
Target presentase kompresi: 0
Path output: /Users/barruadi/Documents/random2/test_output2.jpg
Waktu eksekusi: 300 ms
Ukuran gambar sebelum: 64.63671875
Ukuran gambar sesudah: 40.28515625
Presentase kompresi: 37.674502931044906
Kedalaman pohon: 7
Banyak simpul: 9113

```

Pengujian 4

Gambar sebelum	Gambar setelah
	
<p>Hasil Input dan Output</p> <pre> Path gambar: /Users/barruadi/Documents/random2/test_input4.jpg Metode perhitungan: 4 Threshold: 4 Ukuran blok minimum: 250 Target presentase kompresi: 0 Path output: /Users/barruadi/Documents/random2/test_output4.jpg Waktu eksekusi: 10113 ms Ukuran gambar sebelum: 1539.302734375 Ukuran gambar sesudah: 640.478515625 Presentase kompresi: 58.391646989112104 Kedalaman pohon: 9 Banyak simpul: 69993 </pre>	

Pengujian 5

Gambar sebelum	Gambar setelah
----------------	----------------

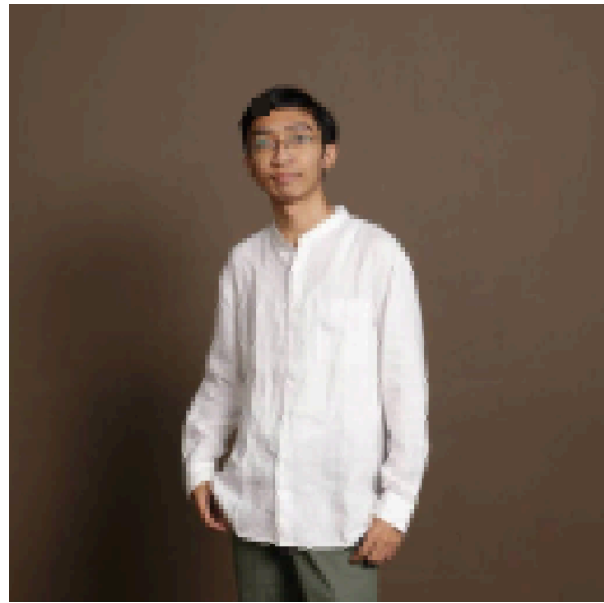


Hasil Input dan Output

```
Path gambar: /Users/barruadi/Documents/random2/test_input.png
Metode perhitungan: 5
Threshold: 0,2
Ukuran blok minimum: 20
Target presentase kompresi: 0
Path output: /Users/barruadi/Documents/random2/test_input5.png
Waktu eksekusi: 688 ms
Ukuran gambar sebelum: 254.3115234375
Ukuran gambar sesudah: 100.58203125
Presentase kompresi: 60.449282875410404
Kedalaman pohon: 8
Banyak simpul: 87381
```

Pengujian 6

Gambar sebelum	Gambar setelah
----------------	----------------



Hasil Input dan Output

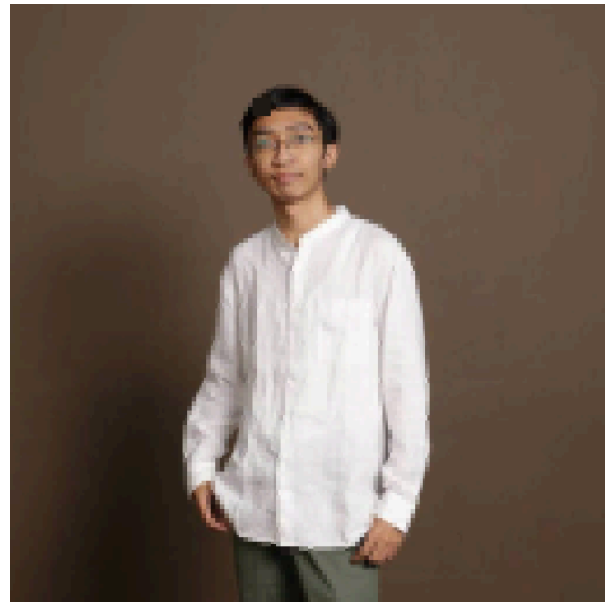
```
Path gambar: /Users/barruadi/Documents/random2/test_input.png
Metode perhitungan: 5
Threshold: 120
Ukuran blok minimum: 40
Target presentase kompresi: 0,5
Path output: /Users/barruadi/Documents/random2/test_output6.png

ERR: Target compression not achieved
Waktu eksekusi: 3831 ms
Ukuran gambar sebelum: 254.3115234375
Ukuran gambar sesudah: 74.03125
Presentase kompresi: 70.88954169306683
Kedalaman pohon: 8
Banyak simpul: 37165
```

Pengujian 7

Gambar sebelum

Gambar setelah



Hasil Input dan Output

```
Path gambar: /Users/barruadi/Documents/random2/test_input.png
Metode perhitungan: 5
Threshold: 1
Ukuran blok minimum: 40
Target presentase kompresi: 0,5
Path output: /Users/barruadi/Documents/random2/test_outputtt.png
ERR: Target compression not achieved
Waktu eksekusi: 3839 ms
Ukuran gambar sebelum: 254.3115234375
Ukuran gambar sesudah: 74.03125
Presentase kompresi: 70.88954169306683
Kedalaman pohon: 8
Banyak simpul: 37165
```

LAMPIRAN

Link Github: https://github.com/barruadi/Tucil2_13523101

Tabel:

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	V	
2. Program berhasil dijalankan	V	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	V	
4. Mengimplementasi seluruh metode perhitungan error wajib	V	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan	V	
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	V	
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar		V
8. Program dan laporan dibuat (kelompok) sendiri	V	