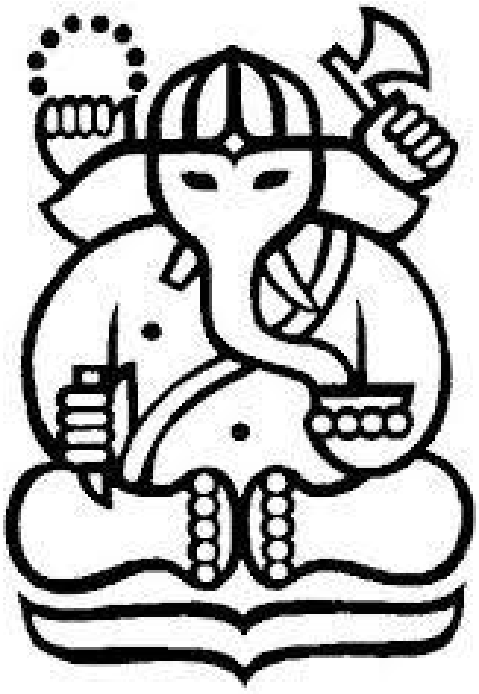
Laporan Tugas Kecil 2
IF2211 Strategi Algoritma 2024/2025
Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:

Varel Tiara (13523008)

Abdullah Farhan (13523042)

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

JL. GANESA 10, BANDUNG 40132

2024

DAFTAR PUSTAKA

BAB I

DESKRIPSI MASALAH

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.

BAB II
ALGORITMA DIVIDE AND CONQUER

2.1 Pendahuluan

Algoritma Divide and Conquer adalah salah satu cara strategi algoritma yang mengandalkan strategi membagi-bagi suatu persoalan yang berukuran besar menjadi beberapa sub-persoalan yang secara struktur sama dengan persoalan semula, namun berukuran lebih kecil. Proses ini dikenal sebagai langkah **Divide**, di mana kita membagi masukan (misalnya larik, matriks, polinom, atau eksponen) berukuran n menjadi r buah sub-masukan dengan ukuran masing-masing $n_1, n_2, ..., n_r$. Idealnya, pembagian ini membuat setiap sub-persoalan berukuran hampir sama agar beban kerja dapat didistribusikan secara merata.

Setelah sub-persoalan terbentuk, langkah **Conquer** dilakukan dengan menyelesaikan setiap sub-persoalan tersebut. Jika ukuran sub-persoalan masih besar (lebih besar dari ambang batas n_0), penyelesaian dilakukan secara rekursif dengan kembali memanggil algoritma Divide and Conquer. Namun, jika ukuran sub-persoalan sudah cukup kecil ($\leq n_0$), kita langsung menyelesaikannya dengan metode yang sederhana atau sudah diketahui.

Langkah **Combine** merupakan tahap penggabungan solusi-solusi dari seluruh sub-persoalan menjadi solusi utuh untuk persoalan semula. Cara penggabungan ini sangat bergantung pada jenis masalahnya masing-masing.

Keunggulan utama Divide and Conquer terletak pada kemampuannya mengeksploitasi struktur rekursif dan paralelisme karena setiap sub-persoalan bersifat independen, tahap Conquer dapat dijalankan secara bersamaan di beberapa prosesor.

2.2 Penjelasan Algoritma Divide and Conquer pada Kompresi Gambar dengan Metode Quadtree

Pendekatan Divide and Conquer pada kompresi gambar dengan metode Quadtree bekerja dengan cara merepresentasikan citra sebagai struktur pohon hierarkis di mana tiap simpul (node) merepresentasikan sebuah blok atau wilayah gambar.

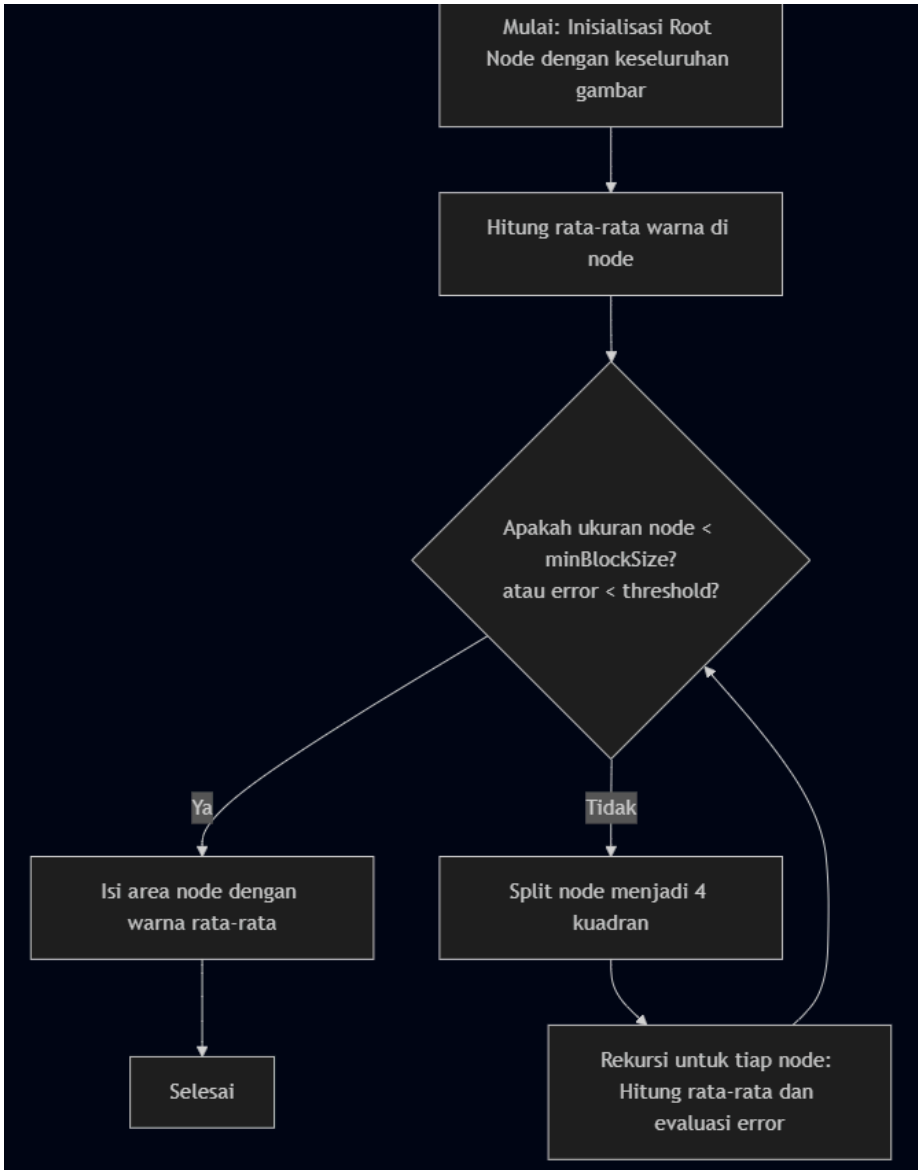
Awalnya, gambar diwakili oleh sebuah simpul akar yang mencakup keseluruhan citra. Kemudian, algoritma menghitung nilai rata-rata warna seluruh pixel di dalam wilayah tersebut dan mengevaluasi error atau perbedaan antara nilai pixel individu dengan rata-rata yang telah dihitung. Jika nilai error melebihi ambang batas tertentu dan ukuran blok masih memadai, simpul tersebut akan dipecah menjadi empat sub-blok secara rekursif. Proses pemecahan (splitting) dilakukan dengan metode rekursif, di mana setiap simpul yang tidak memenuhi kriteria keakuratan diberikan pemecahan lebih lanjut hingga kondisi error sudah di bawah ambang batas atau ukuran blok mencapai ukuran minimum yang telah ditentukan.

Akhirnya, setiap simpul daun yang telah terbentuk akan diwarnai dengan warna rata-rata dari pixel yang terdapat di dalamnya. Dengan demikian, algoritma ini mengurangi kompleksitas citra melalui penyederhanaan warna dalam blok-blok tertentu, memanfaatkan prinsip divide and conquer untuk mencapai kompresi gambar yang efisien dalam hal ruang penyimpanan dan kecepatan pemrosesan, seperti terlihat dalam implementasi pada kelas QuadTree dan QuadTreeNode.

```
Algoritma QuadTree Compression:
1. Inisialisasi:
   - Root node mencakup seluruh gambar
   - Tentukan threshold error dan ukuran blok minimum

2. Divide (Pembagian):
   procedure buildTreeRecursive(node, depth):
       a. Hitung error region menggunakan ErrorMetric
       b. Jika error > threshold DAN ukuran blok > minBlockSize:
           i. Bagi node menjadi 4 sub-region
              (topLeft, topRight, bottomLeft, bottomRight)
          ii. Rekursif proses setiap child node
```

```
       c. Jika tidak:
           i. Simpan sebagai leaf node dengan warna rata-rata

3. Combine (Penggabungan):
   procedure applyColorRecursive(node):
       a. Jika leaf node:
           i. Warnai seluruh region dengan warna rata-rata
       b. Jika bukan leaf:
           i. Rekursif proses semua child node

4. Hasil akhir:
   - Struktur pohon dengan node daun menyimpan warna rata-rata
   - Gambar terkompresi direkonstruksi dari node daun
```

Flowchart


Mulai: Inisialisasi Root Node dengan keseluruhan gambar
Hitung rata-rata warna di node
Apakah ukuran node < minBlockSize? atau error < threshold?
Ya
Tidak
Isi area node dengan warna rata-rata
Split node menjadi 4 kuadran
Selesai
Rekursi untuk tiap node: Hitung rata-rata dan evaluasi error

BAB III

IMPLEMENTASI

3.1 Repositori Github

Berikut tautan repository github kelompok kami untuk Tugas Kecil 2 IF2211 Strategi Algoritma:

https://github.com/Farhanabd05/Tucil2_13523008_13523042

3.2 Spek Bonus yang Dikerjakan

1. Target Persentase Kompresi

Implementasi target kompresi yang telah kami buat berfokus pada pencarian parameter optimal untuk mencapai tingkat kompresi yang diinginkan dengan mempertahankan kualitas gambar yang dapat diterima. Pendekatan utama dalam proses ini adalah penggunaan **QuadTree** untuk membagi gambar ke dalam blok-blok yang lebih kecil, di mana setiap blok akan dikompresi berdasarkan **Error Metric** yang telah dipilih. Jika pengguna tidak menentukan target kompresi (0), maka sistem akan meminta threshold, ukuran minimal blok, dan juga jenis metrik kesalahan yang digunakan, seperti *variance*, *max pixel difference*, *MAD*, *entropy*, atau *SSIM*.

Namun, jika target kompresi diberikan, metode **findOptimalParameters** akan mencari kombinasi terbaik dari ukuran minimal blok dan threshold dengan menguji berbagai parameter dan mengevaluasi tingkat kompresinya dibandingkan dengan target yang telah ditetapkan. Proses ini dilakukan secara iteratif dengan mempersempit rentang pencarian menggunakan langkah-langkah yang lebih kecil untuk mencapai akurasi yang lebih tinggi. Setelah parameter optimal ditemukan, gambar akan dikompresi ulang dengan parameter tersebut, dan hasil akhirnya dievaluasi berdasarkan ukuran file serta tingkat kompresi yang dicapai. Pendekatan ini memastikan bahwa kompresi yang dilakukan tetap optimal dalam menyeimbangkan ukuran file dan kualitas gambar sesuai dengan kebutuhan pengguna.

CompressionController.java

```java
public static class OptimalParameters {
        public double blockSize;
        public double threshold;

        public OptimalParameters(double blockSize, double threshold) {
            this.blockSize = blockSize;
            this.threshold = threshold;
        }

        @Override
        public String toString() {
            return "OptimalParameters { blockSize = " + blockSize + ", threshold = " +
threshold + " }";
        }
    }
```

```java
public static double testCompression(RGBMatrix rgbMatrix, ErrorMetric errorMetric,
                                     double blockSize, double threshold, long
inputFileSize,
                                     String imageFormat) {
        BufferedImage compressedImage = compressImage(rgbMatrix, errorMetric, threshold, (int)
blockSize);
        long compressedSize = getImageSizeInBytes(compressedImage, imageFormat);
        double compressionRate = (1 - ((double) compressedSize / inputFileSize)) * 100.0;
        return compressionRate;
    }
```

```java
public static double getDefaultThreshold(ErrorMetric errorMetric) {
        String metricName = errorMetric.getName().toLowerCase();
        if (metricName.contains("variance")) {
            return 2000;
        } else if (metricName.contains("max pixel difference")) {
            return 15.0;
        } else if (metricName.contains("mad")) {
            return 10.0;
        } else if (metricName.contains("entropy")) {
            return 5.0;
        } else if (metricName.contains("ssim")) {
            return 0.1;
        } else {
            return 10.0;
        }
    }

    public static BufferedImage compressImage(RGBMatrix rgbMatrix, ErrorMetric errorMetric,
                                              double threshold, int blockSize) {
        QuadTree qt = new QuadTree(rgbMatrix.copy(), errorMetric, threshold, blockSize);
        qt.buildTree();
        return OutputHandler.convertToBufferedImage(qt.getRGBMatrix());
    }

    public static long getImageSizeInBytes(BufferedImage image, String formatName) {
        try (ByteArrayOutputStream baos = new ByteArrayOutputStream()) {
            ImageIO.write(image, formatName, baos);
            baos.flush();
            return baos.toByteArray().length;
        } catch (IOException e) {
            e.printStackTrace();
            return 0;
        }
    }
```

```java
public static OptimalParameters findOptimalParameters(RGBMatrix rgbMatrix, ErrorMetric
errorMetric,
                                                       double targetCompressionRate,
String imageFormat,
                                                       boolean useSSIM) {
        BufferedImage originalImage = OutputHandler.convertToBufferedImage(rgbMatrix);
        long inputFileSize = getImageSizeInBytes(originalImage, imageFormat);

        double targetCompression = targetCompressionRate * 100.0;

        int width = rgbMatrix.getWidth();
        int height = rgbMatrix.getHeight();
        int minDimension = Math.min(width, height);
        System.out.println("Target compression: " + targetCompression + "%");

        double maxBlockSize = 1.0;
        while (maxBlockSize * 8 <= minDimension) {
            maxBlockSize *= 8;
        }

        double bestBlockSize = maxBlockSize;
        double bestThreshold = 0.0;
        double closestCompressionRate = 0.0;
        double minDifference = 5.0;

        double detailedStep = useSSIM ? 0.1 : 1.0;
        double detailedTolerance = 1.0;

        boolean foundOptimal = true;
        double blockSize = maxBlockSize;
```

```
while (blockSize >= 4.0 && foundOptimal) {
            System.out.println("Block size: " + blockSize);

            double startThreshold = useSSIM ? 0.1 : 5.0;
            double endThreshold = useSSIM ? 0.9 : 50.0;

            double compressionRateStart = testCompression(rgbMatrix, errorMetric, blockSize,
startThreshold, inputFileSize, imageFormat);
            System.out.println("  Threshold: " + startThreshold + ", Compression: " +
compressionRateStart + "%");

            double diffStart = Math.abs(compressionRateStart - targetCompression);
            System.out.println("  Diff (start): " + diffStart + "%");

            if (diffStart > 15.0) {
                System.out.println("  Skip. Hasil jauh (>15%) dari target.");
                blockSize /= 4;
                continue;
            }

            if (diffStart < minDifference) {
                minDifference = diffStart;
                closestCompressionRate = compressionRateStart;
                bestBlockSize = blockSize;
                bestThreshold = startThreshold;
            }

            double compressionRateEnd = testCompression(rgbMatrix, errorMetric, blockSize,
endThreshold, inputFileSize, imageFormat);
            System.out.println("  End test - Threshold: " + endThreshold + ", Compression: " +
compressionRateEnd + "%");


            double diffEnd = Math.abs(compressionRateEnd - targetCompression);
            if (diffEnd < minDifference) {
                minDifference = diffEnd;
                closestCompressionRate = compressionRateEnd;
                bestBlockSize = blockSize;
                bestThreshold = endThreshold;
            }
```

```
if (diffStart <= 5.0 || diffEnd <= 5.0) {
                System.out.println("  Lakukan pencarian detail threshold (step: " +
detailedStep + ", toleransi: " + detailedTolerance + "%)");
                double currentThreshold = (Math.abs(compressionRateEnd - targetCompression) <
Math.abs(compressionRateStart - targetCompression))
                                        ? endThreshold - detailedStep
                                        : startThreshold + detailedStep;
                while (currentThreshold >= startThreshold && currentThreshold <= endThreshold)
{
                    double currentCompressionRate = testCompression(rgbMatrix, errorMetric,
blockSize, currentThreshold, inputFileSize, imageFormat);
                    double diffCurrent = Math.abs(currentCompressionRate - targetCompression);
                    System.out.println("    Testing - Threshold: " + currentThreshold + ",
Compression: " + currentCompressionRate + "%, Diff: " + diffCurrent + "%");

                    if (diffCurrent < minDifference) {
                        minDifference = diffCurrent;
                        closestCompressionRate = currentCompressionRate;
                        bestBlockSize = blockSize;
                        bestThreshold = currentThreshold;
                        if (diffCurrent <= detailedTolerance) {
                            System.out.println("    Presisi tercapai (diff <= " +
detailedTolerance + "%).");
                            foundOptimal = true;
                            break;
                        }
                    }
                    if ((useSSIM && currentCompressionRate < targetCompression) ||
                        (!useSSIM && currentCompressionRate > targetCompression)) {
                        currentThreshold += detailedStep;
                    } else {
                        currentThreshold -= detailedStep;
                    }
                }
                if (foundOptimal) {
                    break;
                }
            }

            blockSize /= 4;
        }

        System.out.println("Aproksimasi terbaik: " + closestCompressionRate + "% (dengan
error: " + minDifference + "%)");
        System.out.println("  bestThreshold: " + bestThreshold + "%");
        return new OptimalParameters(bestBlockSize, bestThreshold);
    }
```

```java
    public CompressedImage compressWithTarget(RGBMatrix rgbMatrix, ErrorMetric errorMetric,
                                              double targetCompression, String imageFormat)
{
        if (targetCompression == 0) {
            double defaultThreshold = getDefaultThreshold(errorMetric);
            QuadTree qt = new QuadTree(rgbMatrix.copy(), errorMetric, defaultThreshold, 8);
            qt.buildTree();
            BufferedImage compressedImage =
OutputHandler.convertToBufferedImage(qt.getRGBMatrix());
            BufferedImage originalImage = OutputHandler.convertToBufferedImage(rgbMatrix);
            long originalSize = getImageSizeInBytes(originalImage, imageFormat);
            long compressedSize = getImageSizeInBytes(compressedImage, imageFormat);
            double compRate = (1 - ((double) compressedSize / originalSize)) * 100.0;
            return new CompressedImage(compressedImage, compRate, compressedSize,
originalSize, qt);
        }

        boolean useSSIM = errorMetric.getName().toLowerCase().contains("ssim");
        OptimalParameters params = findOptimalParameters(rgbMatrix, errorMetric,
targetCompression, imageFormat, useSSIM);
        System.out.println("Parameter optimal ditemukan: " + params.blockSize + ", " +
params.threshold);
        QuadTree qt = new QuadTree(rgbMatrix.copy(), errorMetric, params.threshold, (int)
params.blockSize);
        qt.buildTree();
        BufferedImage finalCompressedImage =
OutputHandler.convertToBufferedImage(qt.getRGBMatrix());
        BufferedImage originalImage = OutputHandler.convertToBufferedImage(rgbMatrix);
        long originalSize = getImageSizeInBytes(originalImage, imageFormat);
        long finalSize = getImageSizeInBytes(finalCompressedImage, imageFormat);
        double finalCompression = (1 - ((double) finalSize / originalSize)) * 100.0;
        return new CompressedImage(finalCompressedImage, finalCompression, finalSize,
originalSize, qt);
    }
```

CompressionImage.java

```java
import java.awt.image.BufferedImage;
public class CompressedImage {
    private BufferedImage image;
    private double compressionRate;
    private long compressedSizeInBytes;
    private long originalSizeInBytes;
    private QuadTree quadTree;

    public CompressedImage(BufferedImage image, double compressionRate, long
compressedSizeInBytes, long originalSizeInBytes, QuadTree qt) {
        this.image = image;
        this.compressionRate = compressionRate;
        this.compressedSizeInBytes = compressedSizeInBytes;
        this.originalSizeInBytes = originalSizeInBytes;
        this.quadTree = qt;
    }

    public BufferedImage getImage() {
        return image;
    }

    public double getCompressionRate() {
        return compressionRate;
    }

    public long getCompressedSizeInBytes() {
        return compressedSizeInBytes;
    }

    public long getOriginalSizeInBytes() {
        return originalSizeInBytes;
    }

    public QuadTree getQuadTree() {
        return quadTree;
    }
}
```

2. SSIM (Structural Similarity Index Measure)

   Rumus SSIM:

$$SSIM_c(x,y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

Bagian yang dihitung di image:

- Variansi (var1, var2)
- Rata-rata piksel (mu1, mu2)
- Kovariansi (cova)

Output:

- Nilai output untuk region tertentu yang digunakan untuk perbandingan dengan threshold

Implementasi SSIM pada kelas SSIMErrorMetric dirancang untuk mengukur kesamaan struktural antara sebuah region pada gambar dan versinya yang diubah menjadi matriks homogen (monotone) berdasarkan warna rata-rata. Proses dimulai dengan menghitung warna rata-rata dari region yang dianalisis, yang kemudian digunakan untuk membuat matriks seragam yang mewakili area tersebut. Selanjutnya, metode calculateSSIM membagi perhitungan ke dalam tiga kanal warna (merah, hijau, dan biru) dan masing-masing kanal dihitung menggunakan rumus SSIM standar, yang mencakup perhitungan rata-rata, varians, dan kovarians. Konstanta stabilitas (C1 dan C2) digunakan untuk menghindari pembagian oleh nol sehingga nilai SSIM tetap stabil meskipun variasi intensitas kecil terjadi. Hasil SSIM untuk ketiga kanal tersebut kemudian dikombinasikan dengan bobot (0.299 untuk merah, 0.587 untuk hijau, dan 0.114 untuk biru) untuk mendapatkan nilai SSIM akhir. Metode calculateError mengembalikan 1.0 dikurangi nilai SSIM, sehingga nilai error yang tinggi menunjukkan perbedaan struktur yang besar antara region asli dan versi homogennya. Pendekatan ini memungkinkan algoritma kompresi QuadTree untuk menentukan secara lebih presisi kapan sebuah blok harus dibagi, berdasarkan tingkat kesamaan struktural dalam area gambar tersebut.

SSIMErrorMetric

```java
public class SSIMErrorMetric implements ErrorMetric {
    private static final double C1 = 6.5025;  // (0.01 * 255)^2
    private static final double C2 = 58.5225; // (0.03 * 255)^2

    private static final double W_R = 0.299;
    private static final double W_G = 0.587;
    private static final double W_B = 0.114;

    public double calculateSSIMWithMonotoneBlock(RGBMatrix originalMatrix,
    int x, int y, int width, int height, Pixel avgColor) {
        double ssimR = calculateSSIMForChannel(originalMatrix, x, y, width, height, avgColor,
0);
        double ssimG = calculateSSIMForChannel(originalMatrix, x, y, width, height, avgColor,
1);
        double ssimB = calculateSSIMForChannel(originalMatrix, x, y, width, height, avgColor,
2);
        return W_R * ssimR + W_G * ssimG + W_B * ssimB;
    }

    @Override
    public double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height) {
        Pixel avgColor = calculateAverageColor(rgbMatrix, x, y, width, height);

        double ssim = calculateSSIMWithMonotoneBlock(rgbMatrix, x, y, width, height,
avgColor);

        return 1.0 - ssim;
    }

    private Pixel calculateAverageColor(RGBMatrix rgbMatrix, int x, int y, int width, int
height) {
        int totalR = 0, totalG = 0, totalB = 0;
        int count = 0;

        for (int cy = y; cy < y + height && cy < rgbMatrix.getHeight(); cy++) {
            for (int cx = x; cx < x + width && cx < rgbMatrix.getWidth(); cx++) {
                Pixel pixel = rgbMatrix.getPixel(cx, cy);
                totalR += pixel.getR();
                totalG += pixel.getG();
                totalB += pixel.getB();
                count++;
        }
    }

        if (count == 0) return new Pixel(0, 0, 0);

        return new Pixel(totalR / count, totalG / count, totalB / count);
    }
    private double calculateSSIMForChannel(RGBMatrix originalMatrix, int x, int y,
                                           int width, int height, Pixel avgColor, int channel) {
        double sumX = 0;
        double sumXSquare = 0;
        int count = 0;

        double meanY;
        if (channel == 0) {
            meanY = avgColor.getR();
        } else if (channel == 1) {
            meanY = avgColor.getG();
        } else {
            meanY = avgColor.getB();
        }

        for (int cy = y; cy < y + height && cy < originalMatrix.getHeight(); cy++) {
            for (int cx = x; cx < x + width && cx < originalMatrix.getWidth(); cx++) {
                Pixel pixel = originalMatrix.getPixel(cx, cy);
                double pixelX;

                if (channel == 0) {
                    pixelX = pixel.getR();
                } else if (channel == 1) {
                    pixelX = pixel.getG();
                } else {
                    pixelX = pixel.getB();
                }

                sumX += pixelX;
                sumXSquare += pixelX * pixelX;
                count++;
            }
        }

        if (count == 0) return 1.0;

        double meanX = sumX / count;
        double varianceX = (sumXSquare / count) - (meanX * meanX);

        double stdDevX = Math.sqrt(Math.max(0, varianceX));

        double numerator = (2 * meanX * meanY + C1) * C2;
        double denominator = (meanX * meanX + meanY * meanY + C1) * (stdDevX * stdDevX + C2);

        if (denominator == 0) return 1.0;

        return numerator / denominator;
    }

    private double getChannelValue(Pixel pixel, int channel) {
        switch (channel) {
            case 0: return pixel.getR();
            case 1: return pixel.getG();
            case 2: return pixel.getB();
            default: throw new IllegalArgumentException("Invalid channel index: " + channel);
        }
    }

    @Override
    public String getName() {
        return "SSIM (Structural Similarity Index)";
    }
}
```

3. GIF (Animasi Proses Rekonstruksi QuadTree)

Output:

- File gif yang menunjukkan urutan pembuatan quadtree dari depth 0 (atau root node)

Implementasi pembuatan GIF secara dinamis menggunakan kelas GifSequenceWriter. Kelas ini bekerja dengan cara memperoleh *ImageWriter* yang mendukung format GIF melalui API ImageIO, lalu menginisialisasi metadata khusus GIF—seperti pengaturan frame delay, disposal method, dan pengaturan loop—yang disimpan dalam struktur *IIOMetadata*. Selanjutnya, metode writeToSequence mengupdate metadata dengan nilai delay frame sesuai input dan menambahkan *ApplicationExtension* untuk pengaturan looping animasi (misalnya looping tanpa batas apabila kondisi memenuhi). Setiap frame (berupa *BufferedImage*) ditulis ke dalam *ImageOutputStream* sebagai bagian dari rangkaian gambar sehingga menghasilkan file GIF animasi yang utuh. Pendekatan ini memungkinkan pembuat animasi untuk menghasilkan GIF dengan kontrol penuh terhadap pengaturan waktu antar frame dan properti visual lainnya, yang merupakan fitur bonus yang meningkatkan fleksibilitas dalam visualisasi hasil kompresi atau proses lainnya.

GifSequenceWriter.java

```
    protected ImageWriter gifWriter;
    protected ImageWriteParam imageWriteParam;
    protected IIOMetadata imageMetaData;

    public GifSequenceWriter(ImageOutputStream outputStream, int imageType, boolean
loopContinuously) throws IOException {
        gifWriter = getWriter();
        imageWriteParam = gifWriter.getDefaultWriteParam();

        ImageTypeSpecifier imageTypeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
        imageMetaData = gifWriter.getDefaultImageMetadata(imageTypeSpecifier,
imageWriteParam);

        String metaFormatName = imageMetaData.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode) imageMetaData.getAsTree(metaFormatName);

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

```java
    public void writeToSequence(BufferedImage img, int frameDelayMs) throws IOException {
        String metaFormatName = imageMetaData.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode) imageMetaData.getAsTree(metaFormatName);

        IIOMetadataNode graphicsControlExtensionNode = getNode(root,
"GraphicControlExtension");
        graphicsControlExtensionNode.setAttribute("disposalMethod", "none");
        graphicsControlExtensionNode.setAttribute("userInputFlag", "FALSE");
        graphicsControlExtensionNode.setAttribute("transparentColorFlag", "FALSE");
        graphicsControlExtensionNode.setAttribute("delayTime", Integer.toString(frameDelayMs /
10));
        graphicsControlExtensionNode.setAttribute("transparentColorIndex", "0");

        IIOMetadataNode appExtensionsNode = getNode(root, "ApplicationExtensions");
        IIOMetadataNode child = new IIOMetadataNode("ApplicationExtension");
        child.setAttribute("applicationID", "NETSCAPE");
        child.setAttribute("authenticationCode", "2.0");

        int loop = true ? 0 : 1;
        child.setUserObject(new byte[]{0x1, (byte)(loop & 0xFF), (byte)((loop >> 8) & 0xFF)});
        appExtensionsNode.appendChild(child);

        imageMetaData.setFromTree(metaFormatName, root);

        gifWriter.writeToSequence(new IIOImage(img, null, imageMetaData), imageWriteParam);
    }
```

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

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

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

3.3 Struktur Kode Utama

Kode yang telah kami buat terdiri dari beberapa kelas utama, yaitu:

- Main

```java
public static void main(String[] args) {
        InputParser parser = new InputParser();

        try {
            parser.parseInput();
            RGBMatrix rgbMatrix = parser.getRGBMatrix();

            long startTime = System.currentTimeMillis();
            if (!parser.isTargetCompressionSet()) {
                QuadTree quadTree = new QuadTree(rgbMatrix, parser.getErrorMetric(),
parser.getThreshold(), parser.getMinBlockSize());
                quadTree.buildTree();

                long endTime = System.currentTimeMillis();
                long elapsedTime = endTime - startTime;

                OutputHandler.writeImage(quadTree, parser.getOutputPath(),
parser.getInputFile(), elapsedTime);

                if (!parser.getGifPath().isEmpty()) {
                    saveGifEfficiently(quadTree, parser.getGifPath(), 500);
                }
            }
            else {
                CompressionController controller = new CompressionController();
                CompressedImage compressedImage = controller.compressWithTarget(
                    rgbMatrix,
                    parser.getErrorMetric(),
                    parser.getTargetCompression(),
                    parser.getOutputImageFormat()
                );

                long endTime2 = System.currentTimeMillis();
                long elapsedTime2 = endTime2 - startTime;

                BufferedImage compressedImageBuffer = compressedImage.getImage();
                OutputHandler.writeImage2(compressedImageBuffer, parser.getOutputPath(),
parser.getInputFile(), elapsedTime2, compressedImage.getCompressionRate(),
compressedImage.getCompressedSizeInBytes(), compressedImage.getOriginalSizeInBytes());
                System.out.println("Gambar terkompresi telah disimpan.");
                if (!parser.getGifPath().isEmpty()) {
                    saveGifEfficiently(compressedImage.getQuadTree(), parser.getGifPath(),
500);
                }
            }


        } catch (IOException e) {
            System.err.println("[ERROR] Terjadi kesalahan saat membaca file gambar: " +
e.getMessage());
            e.printStackTrace();
        }
    }
```

```java
    private static BufferedImage deepCopy(BufferedImage bi) {
        BufferedImage copy = new BufferedImage(bi.getWidth(), bi.getHeight(), bi.getType());
        Graphics2D g2d = copy.createGraphics();
        g2d.drawImage(bi, 0, 0, null);
        g2d.dispose();
        return copy;
    }

    public static void saveGifEfficiently(QuadTree quadTree, String gifPath, int frameDelay) {
        try {
            CLIUtils.printSectionHeader("CREATING GIF VISUALIZATION");
            System.out.println(CLIUtils.BOLD + "Generating frames and saving GIF..." +
CLIUtils.RESET);

            MemoryEfficientGifWriter efficientWriter = new MemoryEfficientGifWriter(gifPath,
BufferedImage.TYPE_INT_RGB, true);

            int maxDepth = quadTree.getMaxDepth();
            int frameCount = Math.min(maxDepth + 1, 15);
            double depthStep = maxDepth / (double)(frameCount - 1);

            for (int i = 0; i < frameCount; i++) {
                int depth = (int)Math.round(i * depthStep);
                RGBMatrix depthMatrix = new RGBMatrix(quadTree.getRGBMatrix().getWidth(),
quadTree.getRGBMatrix().getHeight());

                // Show current progress
                int percent = (i * 100) / frameCount;
                CLIUtils.printProgressBar(percent);

                quadTree.applyColorsAtDepth(quadTree.getRoot(), depthMatrix, depth, 0);

                BufferedImage frame = OutputHandler.convertToBufferedImage(depthMatrix);
                efficientWriter.writeFrame(frame, frameDelay);

                depthMatrix = null;
                System.gc();
            }

            // Complete the progress bar
            CLIUtils.printProgressBar(100);
            System.out.println();

            efficientWriter.close();
            CLIUtils.printSuccess("GIF animation saved successfully to " + gifPath);
        } catch (IOException e) {
            CLIUtils.printError("Failed to save GIF: " + e.getMessage());
            e.printStackTrace();
        }
    }
```

- QuadTreeNode

```java
    private int x, y;
    private int width, height;
    private Pixel averageColor;
    private QuadTreeNode topLeft, topRight, bottomLeft, bottomRight;
    private boolean isLeaf;

    public QuadTreeNode(int x, int y, int width, int height) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.averageColor = new Pixel(0, 0, 0);
        this.topLeft = null;
        this.topRight = null;
        this.bottomLeft = null;
        this.bottomRight = null;
        this.isLeaf = true;
    }
```

```java
    public void calculateAverageColor(RGBMatrix rgbMatrix) {
        int totalR = 0, totalG = 0, totalB = 0, cnt = 0;
        for (int y = this.y; y < this.y + this.height; y++) {
            for (int x = this.x; x < this.x + this.width; x++) {
                Pixel col = rgbMatrix.getPixel(x, y);
                totalR += col.getR();
                totalG += col.getG();
                totalB += col.getB();
                cnt++;
            }
        }
        if (cnt > 0) {
            this.averageColor = new Pixel(totalR / cnt, totalG / cnt, totalB / cnt);
        } else {
            this.averageColor = new Pixel(0, 0, 0);
        }
    }

    public boolean isLeaf() {
        return isLeaf;
    }
```

```java
    public void split() {
        int halfWidth = this.width / 2;
        int halfHeight = this.height / 2;
        int residualWidth = this.width - halfWidth;
        int residualHeight = this.height - halfHeight;
        this.topLeft = new QuadTreeNode(this.x, this.y, halfWidth, halfHeight);
        this.topRight = new QuadTreeNode(this.x + halfWidth, this.y, residualWidth,
halfHeight);
        this.bottomLeft = new QuadTreeNode(this.x, this.y + halfHeight, halfWidth,
residualHeight);
        this.bottomRight = new QuadTreeNode(this.x + halfWidth, this.y + halfHeight,
residualWidth, residualHeight);

        this.isLeaf = false;
    }

    public int getAverageColorRGB() {
        return (averageColor.getRGB());
    }

    public void setAvgerageColorRGB(int rgb) {
        averageColor.setRGB(rgb);
    }
```

```
    public int getX() { return x; }
    public int getY() { return y; }
    public int getWidth() { return width; }
    public int getHeight() { return height; }
    public Pixel getAverageColor() { return averageColor; }
    public QuadTreeNode getTopLeft() { return topLeft; }
    public QuadTreeNode getTopRight() { return topRight; }
    public QuadTreeNode getBottomLeft() { return bottomLeft; }
    public QuadTreeNode getBottomRight() { return bottomRight; }

    public void setX(int x) { this.x = x; }
    public void setY(int y) { this.y = y; }
    public void setWidth(int width) { this.width = width; }
    public void setHeight(int height) { this.height = height; }
    public void setAvgerageColor(Pixel averageColor) { this.averageColor = averageColor; }
    public void setTopLeft(QuadTreeNode topLeft) { this.topLeft = topLeft; }
    public void setTopRight(QuadTreeNode topRight) { this.topRight = topRight; }
    public void setBottomLeft(QuadTreeNode bottomLeft) { this.bottomLeft = bottomLeft; }
    public void setBottomRight(QuadTreeNode bottomRight) { this.bottomRight = bottomRight; }
```

- QuadTree

```
    private QuadTreeNode root;
    private RGBMatrix rgbMatrix;
    private ErrorMetric errorMetric;
    private double threshold;
    private int minBlockSize;
    private int nodeCount, maxDepth;

    public QuadTree(RGBMatrix rgbMatrix, ErrorMetric errorMetric, double threshold, int
minBlockSize) {
        this.rgbMatrix = rgbMatrix;
        this.errorMetric = errorMetric;
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.nodeCount = 0;
        this.maxDepth = 0;

        this.root = new QuadTreeNode(0, 0, rgbMatrix.getWidth(), rgbMatrix.getHeight());
        this.nodeCount++;
    }

    public void buildTree() {
        buildTreeRecursive(root, 0);
        applyColorToMatrix();
    }
```

```
    private void buildTreeRecursive(QuadTreeNode node, int depth) {
        maxDepth = Math.max(maxDepth, depth);

        node.calculateAverageColor(rgbMatrix);

        if (shouldSplit(node)) {
            node.split();

            buildTreeRecursive(node.getTopLeft(), depth + 1);
            buildTreeRecursive(node.getTopRight(), depth + 1);
            buildTreeRecursive(node.getBottomLeft(), depth + 1);
            buildTreeRecursive(node.getBottomRight(), depth + 1);

            nodeCount += 4;
        }
    }

    private boolean shouldSplit(QuadTreeNode node) {
        if (node.getWidth() * node.getHeight() < minBlockSize) return false;

        double error = errorMetric.calculateError(rgbMatrix, node.getX(), node.getY(),
node.getWidth(), node.getHeight());

        return error > threshold;
    }

    public void buildTree(double currentThreshold, BufferedImage image) {
        buildTreeRecursive(root, 0, currentThreshold, image);
        applyColorToMatrix();
    }
```

```
    private void buildTreeRecursive(QuadTreeNode node, int depth, double currentThreshold,
BufferedImage image) {
        maxDepth = Math.max(maxDepth, depth);

        node.calculateAverageColor(rgbMatrix);

        if (shouldSplit(node, currentThreshold, image)) {
            node.split();

            buildTreeRecursive(node.getTopLeft(), depth + 1, currentThreshold, image);
            buildTreeRecursive(node.getTopRight(), depth + 1, currentThreshold, image);
            buildTreeRecursive(node.getBottomLeft(), depth + 1, currentThreshold, image);
            buildTreeRecursive(node.getBottomRight(), depth + 1, currentThreshold, image);

            nodeCount += 4;
        } else {
            applyColorRecursive(node);
        }
    }

    private boolean shouldSplit(QuadTreeNode node, double currentThreshold, BufferedImage
image) {
        if (node.getWidth() * node.getHeight() < minBlockSize) return false;

        double error = errorMetric.calculateError(rgbMatrix, node.getX(), node.getY(),
node.getWidth(), node.getHeight());

        return error > currentThreshold;
    }
```

```java
    private void applyColorToMatrix() {
        applyColorRecursive(root);
    }

    private void applyColorRecursive(QuadTreeNode node) {
        if (node.isLeaf()) {
            for (int y = node.getY(); y < node.getY() + node.getHeight(); y++) {
                for (int x = node.getX(); x < node.getX() + node.getWidth(); x++) {
                    if (x < rgbMatrix.getWidth() && y < rgbMatrix.getHeight()) {
                        rgbMatrix.setPixel(x, y, node.getAverageColor().getRGB());
                    }
                }
            }
        } else {
            applyColorRecursive(node.getTopLeft());
            applyColorRecursive(node.getTopRight());
            applyColorRecursive(node.getBottomLeft());
            applyColorRecursive(node.getBottomRight());
        }
    }

    public int getNodeCount() {
        return nodeCount;
    }

    public int getMaxDepth() {
        return maxDepth;
    }

    public RGBMatrix getRGBMatrix() {
        return this.rgbMatrix;
    }
```

```
    public void applyColorsAtDepth(QuadTreeNode node, RGBMatrix matrix, int targetDepth, int
currentDepth) {
        if (node == null) return;

        if (currentDepth >= targetDepth || node.isLeaf()) {
            for (int y = node.getY(); y < node.getY() + node.getHeight() && y <
matrix.getHeight(); y++) {
                for (int x = node.getX(); x < node.getX() + node.getWidth() && x <
matrix.getWidth(); x++) {
                    if (node.getAverageColor() == null) matrix.setPixel(x, y, 0);
                    matrix.setPixel(x, y, node.getAverageColor().getRGB());
                }
            }
        }
        else if (currentDepth < targetDepth && !node.isLeaf()) {
            applyColorsAtDepth(node.getTopLeft(), matrix, targetDepth, currentDepth + 1);
            applyColorsAtDepth(node.getTopRight(), matrix, targetDepth, currentDepth + 1);
            applyColorsAtDepth(node.getBottomLeft(), matrix, targetDepth, currentDepth + 1);
            applyColorsAtDepth(node.getBottomRight(), matrix, targetDepth, currentDepth + 1);
        }
}

    public QuadTreeNode getRoot() {
        return this.root;
    }
```

- RGBMatrix

```
    private int width, height;
    private Pixel[] pixels;

    public RGBMatrix(int width, int height) {
        this.width = width;
        this.height = height;
        this.pixels = new Pixel[width * height];
        for (int i = 0; i < pixels.length; i++) {
            pixels[i] = new Pixel(0, 0, 0);
        }
    }

    public void setPixel(int x, int y, int pixel) {
        if (isOutOfBounds(x, y)) {
            throw new IllegalArgumentException("ERROR: Posisi pixel (" + x + ", " + y + ") di
luar batas.");
        }
        pixels[y * width + x].setRGB(pixel);
    }
```

```
    public Pixel getPixel(int x, int y) {
        if (isOutOfBounds(x, y)) {
            throw new IllegalArgumentException("ERROR: Posisi pixel (" + x + ", " + y + ") di
luar batas.");
        }
        return pixels[y * width + x];
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public Pixel[] getPixels() {
        return pixels;
    }

    private boolean isOutOfBounds(int x, int y) {
        return x < 0 || x >= width || y < 0 || y >= height;
    }
```

```as
    public void printRGB() {
        for (int y = 0; y < 3; y++) {
            for (int x = 0; x < 3; x++) {
                System.out.print((getPixel(x, y)) + " ");
                System.out.println();
            }
        }
    }

    public double[] calculateAverageRGB(int x, int y, int width, int height) {
        double sumR = 0, sumG = 0, sumB = 0, cnt = 0;
        for (int y1 = y; y1 < y + height; y1++) {
            for (int x1 = x; x1 < x + width; x1++) {
                if (x1 >= this.getWidth() || y1 >= this.getHeight()) continue;
                Pixel col = this.getPixel(x1, y1);
                sumR += col.getR();
                sumG += col.getG();
                sumB += col.getB();
                cnt++;
            }
        }
        double avgR = sumR / cnt;
        double avgG = sumG / cnt;
        double avgB = sumB / cnt;
        return new double[]{avgR, avgG, avgB};
    }
```


as
public RGBMatrix copy() {
RGBMatrix copy = new RGBMatrix(width, height);
for (int i = 0; i < pixels.length; i++) {
copy.pixels[i] = new Pixel(
pixels[i].getR(),
pixels[i].getG(),
pixels[i].getB()
);
}
return copy;
}

- Pixel

```java
    private int r, g, b;

    public Pixel(int r, int g, int b) {
        this.r = r;
        this.g = g;
        this.b = b;
    }

    public int getR() {
        return r;
    }

    public int getG() {
        return g;
    }

    public int getB() {
        return b;
    }
```

```java
public void setR(int r) {
    this.r = r;
}

public void setG(int g) {
    this.g = g;
}

public void setB(int b) {
    this.b = b;
}

public int getRGB() {
    return (r << 16) | (g << 8) | b;
}

public void setRGB(int rgb) {
    r = (rgb >> 16) & 0xFF;
    g = (rgb >> 8) & 0xFF;
    b = rgb & 0xFF;
}
```

```
    @Override
    public String toString() {
        return "Pixel{" +
                "r=" + r +
                ", g=" + g +
                ", b=" + b +
                '}';
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        Pixel pixel = (Pixel) o;
        return r == pixel.r && g == pixel.g && b == pixel.b;
    }
```

- ErrorMetric

```
public interface ErrorMetric {
    double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height);

    String getName();
}
```

- ErrorMetricFactory

```java
public class ErrorMetricFactory {
    public static ErrorMetric createErrorMetric(int input) {
        return switch (input) {
            case 1 -> new VarianceErrorMetric();
            case 2 -> new MADErrorMetric();
            case 3 -> new MaxPixelDifferenceErrorMetric();
            case 4 -> new EntropyErrorMetric();
            case 5 -> new SSIMErrorMetric();
            default -> throw new IllegalArgumentException("Metode error tidak valid.");
        };
    }
}
```

- MemoryEfficientGifWriter

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

public class MemoryEfficientGifWriter {
    private GifSequenceWriter gifWriter;
    private ImageOutputStream outputStream;

    public MemoryEfficientGifWriter(String outputPath, int imageType, boolean
loopContinuously) throws IOException {
        outputStream = new FileImageOutputStream(new File(outputPath));
        gifWriter = new GifSequenceWriter(outputStream, imageType, loopContinuously);
    }

    public void writeFrame(BufferedImage frame, int frameDelay) throws IOException {
        gifWriter.writeToSequence(frame, frameDelay);
        frame.flush();
    }

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

- VarianceErrorMetric

```java
public class VarianceErrorMetric implements ErrorMetric {

    @Override
    public double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height) {
        double avgR = 0, avgG = 0, avgB = 0, cnt = 0;
        double[] avgRGB = rgbMatrix.calculateAverageRGB(x, y, width, height);
        avgR = avgRGB[0]; avgG = avgRGB[1]; avgB = avgRGB[2];
        double varR = 0, varG = 0, varB = 0;
        for (int y1 = y; y1 < y + height; y1++) {
            for (int x1 = x; x1 < x + width; x1++) {
                Pixel col = rgbMatrix.getPixel(x1, y1);
                double diffR = col.getR() - avgR;
                double diffG = col.getG() - avgG;
                double diffB = col.getB() - avgB;
                varR += diffR * diffR;
                varG += diffG * diffG;
                varB += diffB * diffB;
                cnt++;
            }
        }

        varR /= cnt;
        varG /= cnt;
        varB /= cnt;

        return (varR + varG + varB) / 3.0;
    }

    @Override
    public String getName() {
        return "Variance";
    }
}
```

- MADErrorMetric

```java
public class MADErrorMetric implements ErrorMetric {
    @Override
    public double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height) {
        int cnt = 0;
        double avgR = 0, avgG = 0, avgB = 0;
        double[] avgRGB = rgbMatrix.calculateAverageRGB(x, y, width, height);
        avgR = avgRGB[0]; avgG = avgRGB[1]; avgB = avgRGB[2];
        double madR = 0, madG = 0, madB = 0;
        for (int y1 = y; y1 < y + height; y1++) {
            for (int x1 = x; x1 < x + width; x1++) {
                Pixel col = rgbMatrix.getPixel(x1, y1);
                madR += Math.abs(col.getR() - avgR);
                madG += Math.abs(col.getG() - avgG);
                madB += Math.abs(col.getB() - avgB);
                cnt++;
            }
        }
        madR /= cnt;
        madG /= cnt;
        madB /= cnt;

        return (madR + madG + madB) / 3.0;
    }

    @Override
    public String getName() {
        return "MADErrorMetric";
    }
}
```

- MaxPixelDifferenceErrorMetric

```java
public class MaxPixelDifferenceErrorMetric implements ErrorMetric {

    @Override
    public double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height) {
        double maxR = 0, maxG = 0, maxB = 0, minR = 255, minG = 255, minB = 255;

        for (int y1 = y; y1 < y + height; y1++) {
            for (int x1 = x; x1 < x + width; x1++) {
                if (x1 >= rgbMatrix.getWidth() || y1 >= rgbMatrix.getHeight()) continue;
                Pixel col = rgbMatrix.getPixel(x1, y1);

                maxR = Math.max(maxR, col.getR());
                maxG = Math.max(maxG, col.getG());
                maxB = Math.max(maxB, col.getB());

                minR = Math.min(minR, col.getR());
                minG = Math.min(minG, col.getG());
                minB = Math.min(minB, col.getB());
            }
        }

        double maxDiffR = maxR - minR;
        double maxDiffG = maxG - minG;
        double maxDiffB = maxB - minB;

        return (maxDiffR + maxDiffG + maxDiffB) / 3.0;
    }

    @Override
    public String getName() {
        return "Max Pixel Difference";
    }
}
```

- EntropyErrorMetric

```java
class EntropyErrorMetric implements ErrorMetric {
    @Override
    public double calculateError(RGBMatrix rgbMatrix, int x, int y, int width, int height) {
        int[] freqR = new int[256];
        int[] freqG = new int[256];
        int[] freqB = new int[256];
        int cnt = 0;
        for (int y1 = y; y1 < y + height; y1++) {
            for (int x1 = x; x1 < x + width; x1++) {
                if (x1 >= rgbMatrix.getWidth() || y1 >= rgbMatrix.getHeight()) continue;
                Pixel col = rgbMatrix.getPixel(x1, y1);
                freqR[col.getR()]++;
                freqG[col.getG()]++;
                freqB[col.getB()]++;
                cnt++;
            }
        }

        double entropyR = 0, entropyG = 0, entropyB = 0;
        for (int i = 0; i < 256; i++) {
            double piR = (double) freqR[i] / cnt;
            double piG = (double) freqG[i] / cnt;
            double piB = (double) freqB[i] / cnt;
            if (piR > 0) entropyR -= piR * Math.log(piR) / Math.log(2);
            if (piG > 0) entropyG -= piG * Math.log(piG) / Math.log(2);
            if (piB > 0) entropyB -= piB * Math.log(piB) / Math.log(2);
        }

        return (entropyR + entropyG + entropyB) / 3.0;
    }

    @Override
    public String getName() {
        return "Entropy";
    }
}
```

- OutputHandler

```java
    public static void writeImage(QuadTree quadTree, String outputPath, File inputFile, long
executionTime) throws IOException {
        RGBMatrix rgbMatrix = quadTree.getRGBMatrix();
        int width = rgbMatrix.getWidth();
        int height = rgbMatrix.getHeight();

        BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        Pixel[] pixels = rgbMatrix.getPixels();
        int[] rgbArray = new int[pixels.length];
        for (int i = 0; i < pixels.length; i++) {
            rgbArray[i] = pixels[i].getRGB();
        }
        bufferedImage.setRGB(0, 0, width, height, rgbArray, 0, width);

        String format = getFormatFromPath(outputPath);
        if (format == null) {
            System.err.println("ERROR: Format gambar tidak dikenali.");
            return;
        }

        try (ImageOutputStream ios = ImageIO.createImageOutputStream(new File(outputPath))) {
            ImageIO.write(bufferedImage, format, ios);
        } catch (IOException e) {
            System.err.println("ERROR: Gagal menyimpan gambar ke " + outputPath + ": " +
e.getMessage());
            e.printStackTrace();
        }

        long originalSize = inputFile.length();
        long compressedSize = new File(outputPath).length();

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

        int nodeCount = quadTree.getNodeCount();
        int maxDepth = quadTree.getMaxDepth();
        DecimalFormat df = new DecimalFormat("#.##");

        printCompressionResults(
            df.format(executionTime / 1000.0) + " seconds",
            originalSize + " bytes",
            compressedSize + " bytes",
            df.format(compressionPercentage) + "%",
            String.valueOf(nodeCount),
            String.valueOf(maxDepth)
        );
    }
```

```
    private static void printCompressionResults(String execTime, String origSize, String
compSize, String compPerc, String nodeCount, String maxDepth) {
        String col1Title = "Parameter";
        String col2Title = "Nilai";
        int colWidth1 = 30;
        int colWidth2 = 30;

        String topBorder = "┌" + "─".repeat(colWidth1) + "┬" + "─".repeat(colWidth2) + "┐";
        String midBorder = "├" + "─".repeat(colWidth1) + "┼" + "─".repeat(colWidth2) + "┤";
        String bottomBorder = "└" + "─".repeat(colWidth1) + "┴" + "─".repeat(colWidth2) + "┘";


        System.out.println();
        System.out.println(topBorder);
        System.out.printf("|%" + ((colWidth1 + col1Title.length()) / 2) + "s%" + ((colWidth1 -
col1Title.length() + 1) / 2) + "s", col1Title, "");
        System.out.printf("|%" + ((colWidth2 + col2Title.length()) / 2) + "s%" + ((colWidth2 -
col2Title.length() + 1) / 2) + "s|\n", col2Title, "");
        System.out.println(midBorder);

        printTableRow("Execution time", execTime, colWidth1, colWidth2);
        printTableRow("Original image size", origSize, colWidth1, colWidth2);
        printTableRow("Compressed image size", compSize, colWidth1, colWidth2);
        printTableRow("Compression percentage", compPerc, colWidth1, colWidth2);
        printTableRow("Node count", nodeCount, colWidth1, colWidth2);
        printTableRow("Max depth", maxDepth, colWidth1, colWidth2);

        System.out.println(bottomBorder);
    }
```

```
    private static void printTableRow(String param, String value, int width1, int width2) {
        System.out.printf("| %-"+width1+"s| %-"+width2+"s|\n", param, value);
    }

    private static String getFormatFromPath(String path) {
        int dotIndex = path.lastIndexOf('.');
        if (dotIndex == -1 || dotIndex == path.length() - 1) {
            return null;
        }
        return path.substring(dotIndex + 1);
    }

    public static BufferedImage convertToBufferedImage(RGBMatrix rgbMatrix) {
        int width = rgbMatrix.getWidth();
        int height = rgbMatrix.getHeight();
        BufferedImage bufferedImage = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);

        int[] rgbArray = new int[width * height];
        Pixel[] pixels = rgbMatrix.getPixels();

        for (int i = 0; i < pixels.length; i++) {
            rgbArray[i] = pixels[i].getRGB();
        }

        bufferedImage.setRGB(0, 0, width, height, rgbArray, 0, width);

        rgbArray = null;

        return bufferedImage;
    }
```

```java
    public static void writeImage2(BufferedImage image, String outputPath, File inputFile,
long executionTime, double compressionRate, long compressedSizeInBytes, long
originalSizeInBytes) throws IOException {
        int width = image.getWidth();
        int height = image.getHeight();

        String format = getFormatFromPath(outputPath);
        if (format == null) {
            System.err.println("ERROR: Format gambar tidak dikenali.");
            return;
        }

        try (ImageOutputStream ios = ImageIO.createImageOutputStream(new File(outputPath))) {
            ImageIO.write(image, format, ios);
        } catch (IOException e) {
            System.err.println("ERROR: Gagal menyimpan gambar ke " + outputPath + ": " +
e.getMessage());
            e.printStackTrace();
        }
        DecimalFormat df = new DecimalFormat("#.##");
        System.out.println("\n--- Compression Results ---");
        System.out.println("Execution time: " + df.format(executionTime / 1000.0) + "
seconds");
        System.out.println("Original image size: " + originalSizeInBytes + " bytes");
        System.out.println("Compressed image size: " + compressedSizeInBytes + " bytes");
        System.out.println("Compression percentage: " + compressionRate+ "%");
    }
```

- InputParser

```
    private RGBMatrix rgbMatrix;
    private int height, width;
    private String outputPath;
    private ErrorMetric errorMetric;
    private double threshold;
    private int minBlockSize;
    private File inputFile;
    private String gifPath;
    private double targetCompression;
    private boolean isTargetCompressionSet = false;
    private String outputFormat;

    public InputParser() {}
```

```
    public void parseInput() throws IOException {
        try (Scanner scanner = new Scanner(System.in)) {
            CLIUtils.clearScreen();
            CLIUtils.printLogo();

            // STEP 1: Get input image path
            CLIUtils.printSectionHeader("STEP 1: SELECT INPUT IMAGE");
            System.out.println(CLIUtils.BOLD + "Enter path to image file:" + CLIUtils.RESET);
            System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
            String inputFilePath = scanner.nextLine();

            CLIUtils.simulateLoading("Loading image...", 1500);

            // Load and validate the image
            this.inputFile = new File(inputFilePath);

            // Periksa format file
            String fileExtension = getFileExtension(inputFile);
            if (!fileExtension.equalsIgnoreCase("jpg") &&
!fileExtension.equalsIgnoreCase("jpeg") && !fileExtension.equalsIgnoreCase("png")) {
                CLIUtils.printError("Format file tidak didukung. Hanya JPG, JPEG, dan PNG yang
diperbolehkan.");
                throw new IOException("Unsupported file format");
            }

            BufferedImage image = ImageIO.read(inputFile);
            if (image == null) {
                CLIUtils.printError("Failed to load image. Please ensure the file exists and
is a valid image format.");
                throw new IOException("Cannot read image file");
            }

            this.width = image.getWidth();
            this.height = image.getHeight();
            CLIUtils.printSuccess("Image loaded successfully: " + width + "×" + height + "
pixels");

            // STEP 2: Set output location
            CLIUtils.printSectionHeader("STEP 2: SET OUTPUT LOCATION");
            System.out.println(CLIUtils.BOLD + "Enter path for compressed image:" +
CLIUtils.RESET);
            System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
            this.outputPath = scanner.nextLine();

            // cek jika output format tidak sama dengan input format throw error
            this.outputFormat = outputPath.substring(outputPath.lastIndexOf(".") + 1);
            if (!outputFormat.equalsIgnoreCase(getFileExtension(inputFile))) {
                CLIUtils.printError("Format output tidak sama dengan format input. Silakan
gunakan format yang sama.");
                throw new IOException("Output format mismatch");
            }
            // Convert image to RGB matrix with progress bar
            CLIUtils.printInfo("Converting image to RGB matrix...");
            this.rgbMatrix = new RGBMatrix(width, height);

            int[] rgbArray = image.getRGB(0, 0, width, height, null, 0, width);
            int totalPixels = rgbArray.length;
            for (int i = 0; i < rgbArray.length; i++) {
                rgbMatrix.setPixel(i % width, i / width, rgbArray[i]);

                if (i % (totalPixels / 100) == 0) {
                    CLIUtils.printProgressBar((i * 100) / totalPixels);
                }
            }
            CLIUtils.printProgressBar(100);
            System.out.println();
            CLIUtils.printSuccess("Image converted to RGB matrix");

            // STEP 3: Target compression ratio
            CLIUtils.printSectionHeader("STEP 3: TARGET COMPRESSION");
            System.out.println(CLIUtils.BOLD + "Enter target compression ratio (0 - 1) or
enter 0 to use threshold-based compression:" + CLIUtils.RESET);
            System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
```

```
            while (true) {
                try {
                    this.targetCompression = scanner.nextDouble();
                    scanner.nextLine(); // Consume newline

                    if (this.targetCompression < 0 || this.targetCompression > 1) {
                        CLIUtils.printError("Invalid target compression ratio. Please enter a
value between 0 and 1.");
                        System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                        continue;
                    }
                    break;
                } catch (Exception e) {
                    scanner.nextLine(); // Clear the scanner
                    CLIUtils.printError("Input must be a number. Please try again.");
                    System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                }
            }

            if (this.targetCompression > 0) {
                this.isTargetCompressionSet = true;
                CLIUtils.printSuccess("Target compression ratio set to " +
this.targetCompression);
            } else {
                CLIUtils.printInfo("Using threshold-based compression instead of target
ratio");
            }

            // STEP 4: Compression parameters
            CLIUtils.printSectionHeader("STEP 4: COMPRESSION PARAMETERS");
            displayErrorMetrics();

            // Get and validate error metric selection
            int errorMetricChoice;
            System.out.println(CLIUtils.BOLD + "Select error metric (1-5):" + CLIUtils.RESET);
            System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);

            while (true) {
                try {
                    errorMetricChoice = scanner.nextInt();
                    if (errorMetricChoice < 1 || errorMetricChoice > 5) {
                        CLIUtils.printError("Please enter a number between 1 and 5.");
                        System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                        continue;
                    }
                    break;
                } catch (Exception e) {
                    scanner.nextLine(); // Clear the scanner
                    CLIUtils.printError("Input must be a number. Please try again.");
                    System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                }
            }

            errorMetric = ErrorMetricFactory.createErrorMetric(errorMetricChoice);
            CLIUtils.printSuccess("Selected error metric: " + errorMetric.getName());
```

```
            // If not using target compression, get threshold and min block size
            if (!this.isTargetCompressionSet) {
                // Get and validate threshold based on selected error metric
                this.threshold = getValidThreshold(scanner, errorMetricChoice);

                // Get and validate minimum block size
                this.minBlockSize = getValidMinBlockSize(scanner);
            } else {
                scanner.nextLine(); // Consume newline
            }

            // STEP 5: GIF visualization path
            CLIUtils.printSectionHeader("STEP 5: GIF VISUALIZATION");
            System.out.println(CLIUtils.BOLD + "Enter path for GIF animation (leave empty to
skip):" + CLIUtils.RESET);
            System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
            this.gifPath = scanner.nextLine();

            CLIUtils.clearScreen();
            // Display summary and prepare for compression
            CLIUtils.printSectionHeader("READY TO COMPRESS");
            CLIUtils.printInfo("Image: " + inputFile.getName() + " (" + width + "×" + height +
")");
            CLIUtils.printInfo("Error metric: " + errorMetric.getName());

            if (!isTargetCompressionSet) {
                CLIUtils.printInfo("Error threshold: " + threshold);
                CLIUtils.printInfo("Minimum block size: " + minBlockSize);
            } else {
                CLIUtils.printInfo("Target compression ratio: " + targetCompression);
            }

            if (!gifPath.isEmpty()) {
                CLIUtils.printInfo("GIF animation will be created at: " + gifPath);
            } else {
                CLIUtils.printWarning("GIF animation skipped");
            }

            System.out.println(CLIUtils.BOLD + CLIUtils.GREEN + "\nStarting compression in 3
seconds..." + CLIUtils.RESET);
            try {
                Thread.sleep(3000);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
            }
        }
    }
```

```
    private double getValidThreshold(Scanner scanner, int errorMetricChoice) {
        double thresholdValue;
        scanner.nextLine(); // Consume newline

        System.out.println(CLIUtils.BOLD + "Enter error threshold value:" + CLIUtils.RESET);
        System.out.println(CLIUtils.CYAN + "Note - Valid ranges for each metric:" +
CLIUtils.RESET);
        System.out.println("1. Variance: (0 - 16256.25)");
        System.out.println("2. Mean Absolute Deviation: (0 - 127.5)");
        System.out.println("3. Max Pixel Difference: (0 - 255)");
        System.out.println("4. Entropy: (0 - 8)");
        System.out.println("5. SSIM: (0 - 1)");
        System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);

        while (true) {
            try {
                thresholdValue = scanner.nextDouble();
                scanner.nextLine(); // Consume newline

                boolean isValid = true;

                // Validate threshold based on selected error metric
                switch (errorMetricChoice) {
                    case 1: // Variance
                        if (thresholdValue < 0 || thresholdValue > 16256.25) {
                            CLIUtils.printError("Invalid range! Threshold for Variance must be
between 0 and 16256.25");
                            isValid = false;
                        }
                        break;
                    case 2: // MAD
                        if (thresholdValue < 0 || thresholdValue > 127.5) {
                            CLIUtils.printError("Invalid range! Threshold for MAD must be
between 0 and 127.5");
                            isValid = false;
                        }
                        break;
                    case 3: // Max Pixel Difference
                        if (thresholdValue < 0 || thresholdValue > 255) {
                            CLIUtils.printError("Invalid range! Threshold for Max Pixel
Difference must be between 0 and 255");
                            isValid = false;
                        }
                        break;
                    case 4: // Entropy
                        if (thresholdValue < 0 || thresholdValue > 8) {
                            CLIUtils.printError("Invalid range! Threshold for Entropy must be
between 0 and 8");
                            isValid = false;
                        }
                        break;
                    case 5: // SSIM
                        if (thresholdValue < 0 || thresholdValue > 1) {
                            CLIUtils.printError("Invalid range! Threshold for SSIM must be
between 0 and 1");
                            isValid = false;
                        }
                        break;
                }

                if (isValid) {
                    CLIUtils.printSuccess("Threshold value set to: " + thresholdValue);
                    return thresholdValue;
                } else {
                    System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                }
            } catch (Exception e) {
                scanner.nextLine(); // Clear the scanner
                CLIUtils.printError("Input must be a number. Please try again.");
                System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
            }
        }
    }
```

```
    private int getValidMinBlockSize(Scanner scanner) {
        int minSize;

        System.out.println(CLIUtils.BOLD + "Enter minimum block size:" + CLIUtils.RESET);
        System.out.println(CLIUtils.CYAN + "Note: Size must be positive and not larger than
the original image dimensions" + CLIUtils.RESET);
        System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);

        while (true) {
            try {
                minSize = scanner.nextInt();
                scanner.nextLine(); // Consume newline

                if (minSize <= 0) {
                    CLIUtils.printError("Minimum size must be positive. Please try again.");
                    System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                } else if (minSize > height * width) {
                    CLIUtils.printError("Minimum size cannot be larger than the image size ("
+ (height * width) + " pixels). Please try again.");
                    System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
                } else {
                    CLIUtils.printSuccess("Minimum block size set to: " + minSize);
                    return minSize;
                }
            } catch (Exception e) {
                scanner.nextLine(); // Clear the scanner
                CLIUtils.printError("Input must be a number. Please try again.");
                System.out.print(CLIUtils.GREEN + ">> " + CLIUtils.RESET);
            }
        }
    }
```

```
    public static void displayErrorMetrics() {
        System.out.println(CLIUtils.BOLD + "Available Error Metrics:" + CLIUtils.RESET);
        System.out.println(CLIUtils.CYAN + "1. " + CLIUtils.RESET + "Variance (Standard
deviation of pixel values)");
        System.out.println(CLIUtils.CYAN + "2. " + CLIUtils.RESET + "Mean Absolute Deviation
(Average difference from mean)");
        System.out.println(CLIUtils.CYAN + "3. " + CLIUtils.RESET + "Max Pixel Difference
(Maximum difference between pixels)");
        System.out.println(CLIUtils.CYAN + "4. " + CLIUtils.RESET + "Entropy (Information
content measure)");
        System.out.println(CLIUtils.CYAN + "5. " + CLIUtils.RESET + "SSIM (Structural
Similarity Index)");
    }

    // Getters
    public int getHeight() { return height; }
    public int getWidth() { return width; }
    public RGBMatrix getRGBMatrix() { return rgbMatrix; }
    public String getOutputPath() { return outputPath; }
    public ErrorMetric getErrorMetric() { return errorMetric; }
    public double getThreshold() { return threshold; }
    public int getMinBlockSize() { return minBlockSize; }
    public File getInputFile() { return inputFile; }
    public String getGifPath() { return gifPath; }
    public double getTargetCompression() { return targetCompression; }
    public boolean isTargetCompressionSet() { return isTargetCompressionSet; }
    public String getOutputImageFormat() { return outputFormat; }
    // Fungsi untuk mendapatkan ekstensi file
    private String getFileExtension(File file) {
        String fileName = file.getName();
        int extensionIndex = fileName.lastIndexOf(".");
        if (extensionIndex > 0) {
            return fileName.substring(extensionIndex + 1).toLowerCase();
        } else {
            return "";
        }
    }
```

- CLI

```java
public class CLIUtils {
    // ANSI color codes
    public static final String RESET = "\u001B[0m";
    public static final String BLACK = "\u001B[30m";
    public static final String RED = "\u001B[31m";
    public static final String GREEN = "\u001B[32m";
    public static final String YELLOW = "\u001B[33m";
    public static final String BLUE = "\u001B[34m";
    public static final String PURPLE = "\u001B[35m";
    public static final String CYAN = "\u001B[36m";
    public static final String WHITE = "\u001B[37m";

    // Background colors
    public static final String BG_BLACK = "\u001B[40m";
    public static final String BG_RED = "\u001B[41m";
    public static final String BG_GREEN = "\u001B[42m";
    public static final String BG_YELLOW = "\u001B[43m";
    public static final String BG_BLUE = "\u001B[44m";
    public static final String BG_PURPLE = "\u001B[45m";
    public static final String BG_CYAN = "\u001B[46m";
    public static final String BG_WHITE = "\u001B[47m";

    // Text styles
    public static final String BOLD = "\u001B[1m";
    public static final String UNDERLINE = "\u001B[4m";


    public static void clearScreen() {
        System.out.print("\033c");       // Reset terminal
        System.out.print("\033[2J");     // Clear entire screen
        System.out.print("\033[H");      // Move cursor to top-left corner
        System.out.print("\033[3J");     // Clear scrollback buffer (supported in
some terminals)
        System.out.flush();              // Ensure all commands are sent immediately

    }


    public static void printSectionHeader(String title) {
        int boxWidth = 45;
        System.out.println();
        System.out.println(BOLD + BLUE + "┌" + "─".repeat(boxWidth - 2) + "┐" +
RESET);
        System.out.println(BOLD + BLUE + "| " + YELLOW + title +
                " ".repeat(Math.max(0, boxWidth - title.length() - 4)) + BLUE + "|"
+ RESET);
        System.out.println(BOLD + BLUE + "└" + "─".repeat(boxWidth - 2) + "┘" +
RESET);
    }
```

```
public static void printProgressBar(int percent) {
        final int width = 50; // Progress bar width
        int completed = (int)((double)percent / 100 * width);

        System.out.print("\r[");
        for (int i = 0; i < width; i++) {
            if (i < completed) {
                System.out.print(GREEN + "" + RESET);
            } else {
                System.out.print(" ");
            }
        }
        System.out.print("] " + percent + "%");
    }

    public static void printSuccess(String message) {
        System.out.println(GREEN + "✓ " + message + RESET);
    }

    public static void printError(String message) {
        System.out.println(RED + "✗ " + message + RESET);
    }

    public static void printInfo(String message) {
        System.out.println(BLUE + "ℹ " + message + RESET);
    }

    public static void printWarning(String message) {
        System.out.println(YELLOW + "⚠ " + message + RESET);
    }

    public static void simulateLoading(String message, int durationMs) {
        String[] frames = {"⠋", "⠙", "⠹", "⠸", "⠼", "⠴", "⠦", "⠧", "⠇", "⠏"};
        long startTime = System.currentTimeMillis();
        long endTime = startTime + durationMs;
        int frameIndex = 0;

        while (System.currentTimeMillis() < endTime) {
            System.out.print("\r" + CYAN + frames[frameIndex] + " " + message +
RESET);
            frameIndex = (frameIndex + 1) % frames.length;
            try {
                Thread.sleep(80);
            } catch (InterruptedException e) {
                Thread.currentThread().interrupt();
                break;
            }
        }
        System.out.println();
    }
}
```

BAB IV

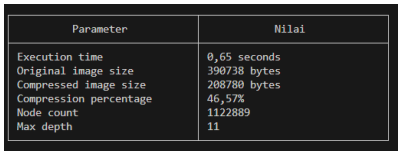
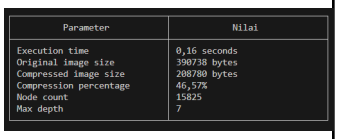
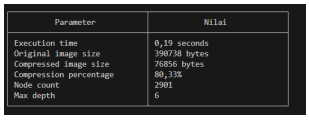
EKSPERIMEN

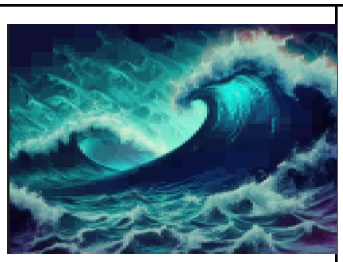
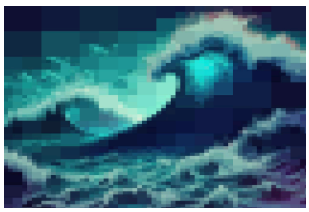
4.1 Hasil Pengujian

a. Kasus Eksperimen 1 JPG

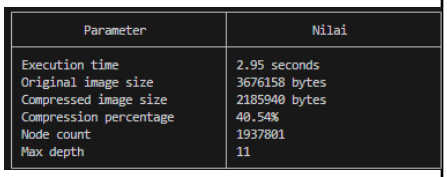
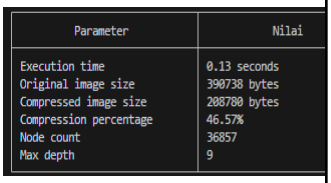
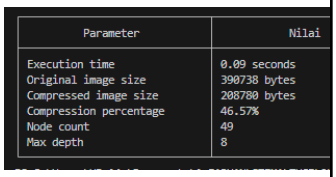
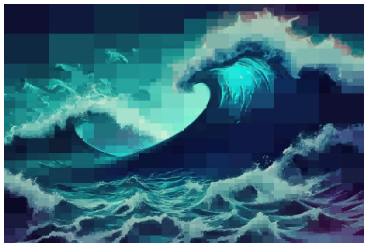
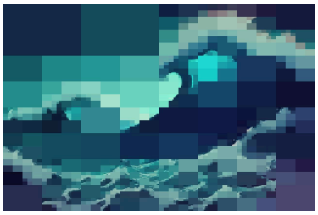
test/tc/tc1.jpg

i. Variance

threshold min block size	10 1	100 200	500 1000
Terminal Output	 Parameter | Nilai Execution time | 0,65 seconds Original image size | 390738 bytes Compressed image size | 208780 bytes Compression percentage | 46,57% Node count | 1122889 Max depth | 11	 Parameter | Nilai Execution time | 0,16 seconds Original image size | 390738 bytes Compressed image size | 208780 bytes Compression percentage | 46,57% Node count | 15825 Max depth | 7	 Parameter | Nilai Execution time | 0,19 seconds Original image size | 390738 bytes Compressed image size | 76856 bytes Compression percentage | 80,33% Node count | 2901 Max depth | 6

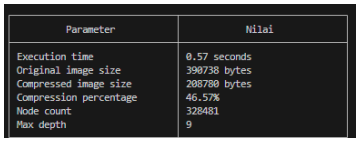
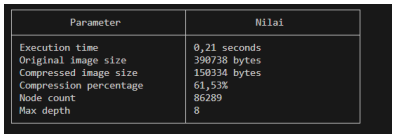
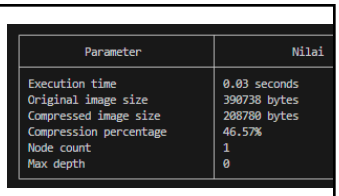
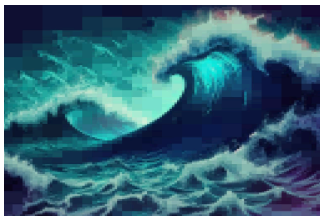
Gambar			
Path Gif	test/sol/tc1_1.gif https://github.com/Farhanabd05/Tucil2_13523008_13523042/blob/main/test/sol/tc1_1.gif	test/sol/tc1_2.gif https://github.com/Farhanabd05/Tucil2_13523008_13523042/blob/main/test/sol/tc1_2.gif	test/sol/tc1_3.gif https://github.com/Farhanabd05/Tucil2_13523008_13523042/blob/main/test/sol/tc1_3.gif

ii. MAD

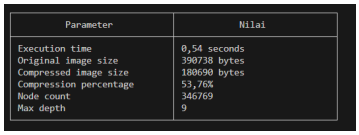
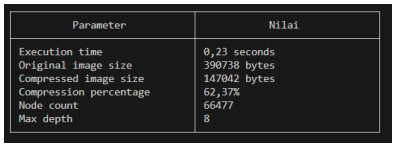
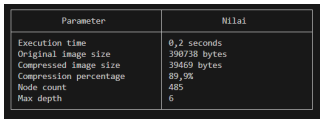
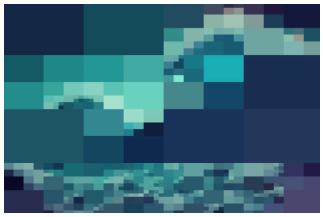
threshold min block size	5 8	15 16	45 64
Terminal Output			
Gambar			

iii. MPD

threshold min block size	2 10	4 25	8 50

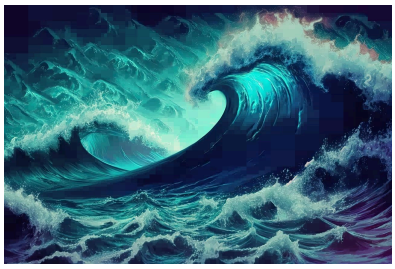
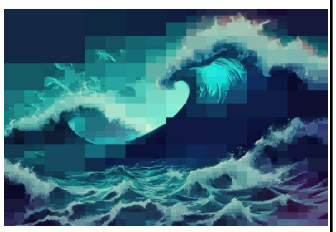
Terminal Output			
Gambar			

iv. Entropy

threshold min block size	1 10	4 50	7 100
Terminal Output			
Gambar			

v. SSIM

threshold min block size target	0,1 1 0	0.5 1 0	0.9 16 0

Terminal Output	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0.3 seconds</td></tr><tr><td>Original image size</td><td>211682 bytes</td></tr><tr><td>Compressed image size</td><td>209629 bytes</td></tr><tr><td>Compression percentage</td><td>0.97%</td></tr><tr><td>Node count</td><td>1250025</td></tr><tr><td>Max depth</td><td>11</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0.23 seconds</td></tr><tr><td>Original image size</td><td>211682 bytes</td></tr><tr><td>Compressed image size</td><td>194062 bytes</td></tr><tr><td>Compression percentage</td><td>8.32%</td></tr><tr><td>Node count</td><td>565765</td></tr><tr><td>Max depth</td><td>11</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0.15 seconds</td></tr><tr><td>Original image size</td><td>211682 bytes</td></tr><tr><td>Compressed image size</td><td>100032 bytes</td></tr><tr><td>Compression percentage</td><td>52.74%</td></tr><tr><td>Node count</td><td>23777</td></tr><tr><td>Max depth</td><td>9</td></tr></table>
Gambar			

b. Kasus Eksperimen 1 PNG

Gambar Asli:

i. Variance

threshold minblocksize	10 1	100 200	500 1000
Terminal Output	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0,65 seconds</td></tr><tr><td>Original image size</td><td>390738 bytes</td></tr><tr><td>Compressed image size</td><td>208780 bytes</td></tr><tr><td>Compression percentage</td><td>46,57%</td></tr><tr><td>Node count</td><td>1122889</td></tr><tr><td>Max depth</td><td>11</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0.43 seconds</td></tr><tr><td>Original image size</td><td>15933389 bytes</td></tr><tr><td>Compressed image size</td><td>724784 bytes</td></tr><tr><td>Compression percentage</td><td>95.45%</td></tr><tr><td>Node count</td><td>27953</td></tr><tr><td>Max depth</td><td>8</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>0.39 seconds</td></tr><tr><td>Original image size</td><td>15933389 bytes</td></tr><tr><td>Compressed image size</td><td>284659 bytes</td></tr><tr><td>Compression percentage</td><td>98.21%</td></tr><tr><td>Node count</td><td>4329</td></tr><tr><td>Max depth</td><td>7</td></tr></table>

Gambar			

ii. MAD

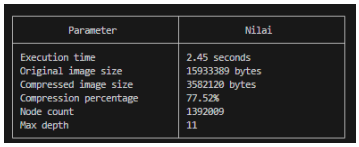
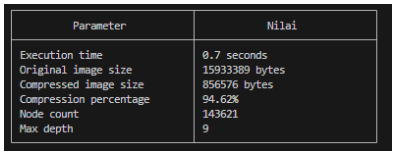
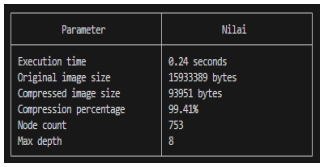
threshold minblocksize	5 8	15 16	45 64
Terminal Output	Parameter | Nilai Execution time | 0.47 seconds Original image size | 15933389 bytes Compressed image size | 2435518 bytes Compression percentage | 84.71% Node count | 700329 Max depth | 11	Parameter | Nilai Execution time | 0.42 seconds Original image size | 15933389 bytes Compressed image size | 506943 bytes Compression percentage | 96.82% Node count | 84937 Max depth | 10	Parameter | Nilai Execution time | 0.27 seconds Original image size | 15933389 bytes Compressed image size | 54984 bytes Compression percentage | 99.65% Node count | 197 Max depth | 9
Gambar			

iii. MPD

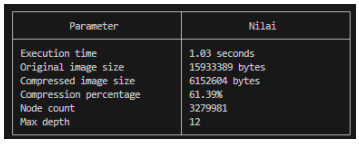
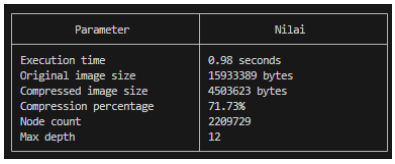
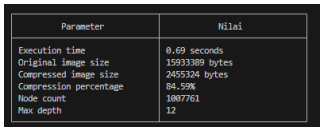
threshold min block size	2 10	127 50	255 100
Terminal Output	Parameter | Nilai Execution time | 0.71 seconds Original image size | 15933389 bytes Compressed image size | 3815380 bytes Compression percentage | 76.05% Node count | 1608957 Max depth | 11	Parameter | Nilai Execution time | 0.37 seconds Original image size | 15933389 bytes Compressed image size | 348872 bytes Compression percentage | 97.81% Node count | 23453 Max depth | 9	Parameter | Nilai Execution time | 0.11 seconds Original image size | 15933389 bytes Compressed image size | 29373 bytes Compression percentage | 99.82% Node count | 1 Max depth | 0

Gambar			

iv. Entropy

threshold min block size	2 10	4 50	7 100
Terminal Output	 Parameter | Nilai Execution time | 2.45 seconds Original image size | 15933389 bytes Compressed image size | 3582120 bytes Compression percentage | 77.52% Node count | 1392089 Max depth | 11	 Parameter | Nilai Execution time | 0.7 seconds Original image size | 15933389 bytes Compressed image size | 856576 bytes Compression percentage | 94.62% Node count | 143621 Max depth | 9	 Parameter | Nilai Execution time | 0.24 seconds Original image size | 15933389 bytes Compressed image size | 93951 bytes Compression percentage | 99.41% Node count | 753 Max depth | 8
Gambar			

v. SSIM

threshold min block size target	0,1 1 0	0.2 1 0	0.5 2 0
Terminal Output	 Parameter | Nilai Execution time | 1.03 seconds Original image size | 15933389 bytes Compressed image size | 6152604 bytes Compression percentage | 61.39% Node count | 3279981 Max depth | 12	 Parameter | Nilai Execution time | 0.98 seconds Original image size | 15933389 bytes Compressed image size | 4503623 bytes Compression percentage | 71.73% Node count | 2209729 Max depth | 12	 Parameter | Nilai Execution time | 0.69 seconds Original image size | 15933389 bytes Compressed image size | 2455324 bytes Compression percentage | 84.59% Node count | 1007761 Max depth | 12

Gambar			

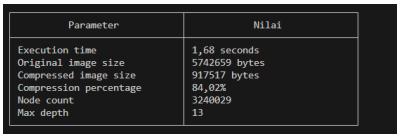
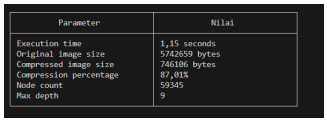
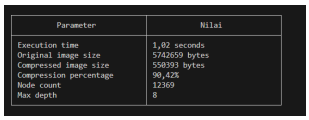
c. Kasus Eksperimen 1 JPEG

Gambar Asli:

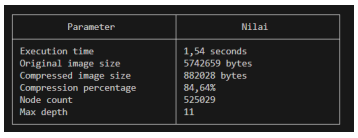
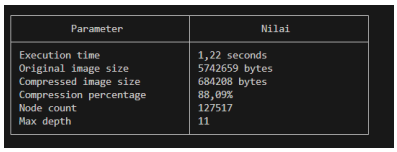
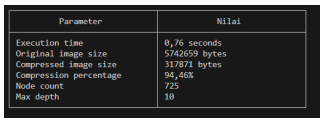
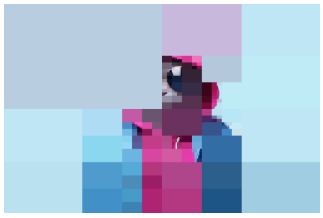
Hasil Kompresi:

i. Variance

threshold min block size	10 1	100 200	500 1000

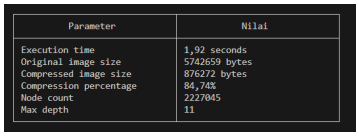
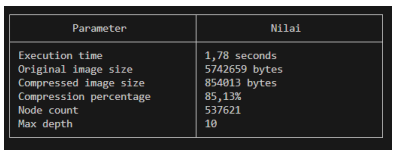
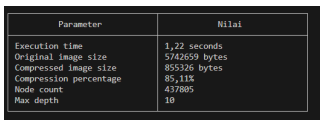
Terminal Output			
Gambar			
Path Gif (Sebagai contoh saja kalau work)	test/sol/tc1_1.gif	test/sol/tc1_2.gif	test/sol/tc1_3.gif

ii. MAD

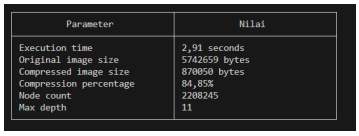
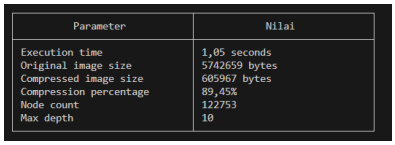
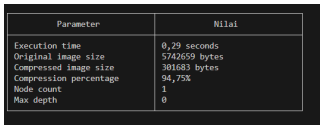
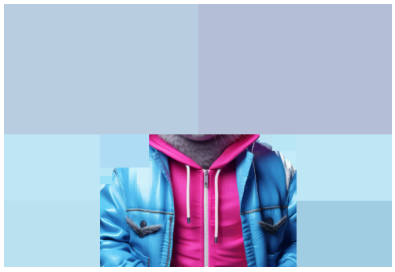
threshold min block size	5 8	15 16	45 64
Terminal Output			
Gambar			

iii. MPD

threshold min block size	2 10	4 25	8 50

Terminal Output			
Gambar			

iv. Entropy

threshold min block size	1 10	4 50	6 50
Terminal Output			
Gambar			

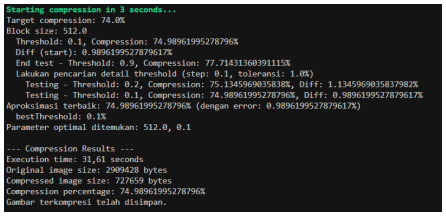
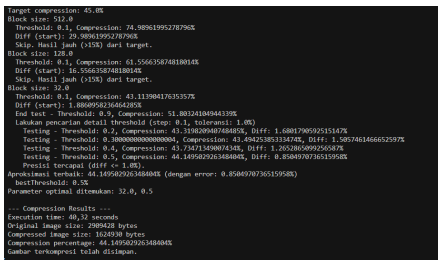
v. SSIM

threshold min block size target	0,1 1 0	0.5 1 0	0.9 16 0

Terminal Output	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>1.93 seconds</td></tr><tr><td>Original image size</td><td>900680 bytes</td></tr><tr><td>Compressed image size</td><td>913730 bytes</td></tr><tr><td>Compression percentage</td><td>-1.45%</td></tr><tr><td>Node count</td><td>3821333</td></tr><tr><td>Max depth</td><td>13</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>1.33 seconds</td></tr><tr><td>Original image size</td><td>900680 bytes</td></tr><tr><td>Compressed image size</td><td>884265 bytes</td></tr><tr><td>Compression percentage</td><td>1.82%</td></tr><tr><td>Node count</td><td>1365581</td></tr><tr><td>Max depth</td><td>13</td></tr></table>	<table><tr><th>Parameter</th><th>Nilai</th></tr><tr><td>Execution time</td><td>1.11 seconds</td></tr><tr><td>Original image size</td><td>900680 bytes</td></tr><tr><td>Compressed image size</td><td>617620 bytes</td></tr><tr><td>Compression percentage</td><td>31.43%</td></tr><tr><td>Node count</td><td>88917</td></tr><tr><td>Max depth</td><td>11</td></tr></table>
Gambar			

d. Kasus Target Compression

target	0,74	0,46

Terminal Output		
Gambar		

BAB IV

ANALISIS

Algoritma Divide and Conquer untuk kompresi gambar menggunakan metode Quadtree bekerja dengan cara membagi gambar secara rekursif menjadi empat kuadran hingga tiap blok memenuhi kondisi pemberhentian tertentu, yaitu ketika error sudah di bawah ambang batas (threshold) atau ukuran blok telah mencapai batas minimum. Pendekatan yang digunakan oleh Quadtree adalah pendekatan top-down dimana proses dimulai dari gambar utuh, lalu dipecah secara bertahap sehingga setiap blok hasil pembagian dapat direpresentasikan dengan nilai rata-rata pikselnya.

Secara waktu, pembangunan Quadtree memiliki kompleksitas O(n log n) di mana n adalah jumlah total piksel, karena setiap piksel diproses setidaknya sekali dan pada setiap level seluruh gambar dihitung ulang error-nya dengan kedalaman maksimum tree sekitar $\log_4(n)$ karena setiap node dibagi menjadi empat anak.

Metode perhitungan error juga memiliki kompleksitas tersendiri, bergantung pada jenis error metric yang digunakan (misalnya Variance memerlukan dua pass melalui seluruh pixel pada blok, MAD (Mean Absolute Deviation) atau Max Pixel Difference atau Entropy hanya memerlukan satu pass untuk menghitung selisih rata-rata yang absolut (MAD), untuk menghitung perbedaan pada pixel secara maximum (MPD), dan untuk menghitung frekuensi dan kemudian perhitungan entropy, sedangkan SSIM memerlukan beberapa pass untuk menghitung mean, variance, dan covariance).

Sedangkan, proses rekonstruksi gambar dari Quadtree memiliki kompleksitas O(n), karena setiap piksel dikunjungi tepat satu kali.

Dari kompleksitas ruang, struktur Quadtree dalam kasus terburuk memerlukan O(n) ruang, karena jumlah node dalam pohon lengkap bisa mencapai sekitar 4n/3, ditambah dengan ruang tambahan O(w×h) jika metode error (seperti SSIM) membuat gambar sementara.

Implementasi ini menggunakan strategi pemberhentian ganda, threshold error dan ukuran blok minimum, yang dirancang untuk menjaga keseimbangan antara efisiensi komputasi dan

kualitas visual gambar, sehingga sangat cocok digunakan dalam aplikasi kompresi gambar yang membutuhkan penyesuaian kualitas secara dinamis.

BAB V

KESIMPULAN, REFLEKSI

5.1 Kesimpulan

Melalui pengerjaan tugas kecil ini, kami telah berhasil mengimplementasikan algoritma Divide and Conquer dengan teknik rekursi untuk melakukan kompresi gambar dengan metode Quadtree. Dalam tugas ini, kami mengembangkan sebuah program yang dapat meminta input path gambar dan parameter yang dibutuhkan, melakukan kompresi gambar sesuai dengan parameter, membuat gif dari gambar yang telah dikompresi, serta membuat kompresi gambar sesuai dengan target.

5.2 Refleksi

Pengerjaan tugas kecil ini memberikan pengalaman yang sangat berharga dalam mengaplikasikan algoritma Divide and Conquer. Tantangan utama yang telah kami hadapi adalah memastikan bahwa algoritma berjalan dengan baik dan efisien. Kami juga belajar mengenai pentingnya validasi input dan penangan error yang baik, terutama saat membaca file dan melakukan kompresi pada gambar. Pengalaman ini tentunya akan sangat berguna dalam proyek-proyek teknologi yang lebih kompleks di masa depan.

Untuk Tuhan, Bangsa, dan Almamater. Hidup Informatika!

BAB VI

LAMPIRAN

6.1 Tautan Repository

Link repository dari Tugas Kecil 2 IF2211 Strategi Algoritma adalah sebagai berikut:

https://github.com/Farhanabd05/Tucil2_13523008_13523042

6.2 Daftar Referensi

Munir, Rinaldi. 2025. "Algoritma Divide and Conquer (Bagian 1)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

Munir, Rinaldi. 2025. "Algoritma Divide and Conquer (Bagian 2)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf)

Munir, Rinaldi. 2025. "Algoritma Divide and Conquer (Bagian 3)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf)

Munir, Rinaldi. 2025. "Algoritma Divide and Conquer (Bagian 4)" (https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian4.pdf)

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	✓	
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	✓	