

**LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA**

Kompresi Gambar Dengan Metode Quadtree



Disusun Oleh:
Raka Daffa Iftikhaar (13523018)
Kefas Kurnia Jonathan (135230113)

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

BAB 1

Teori Singkat

Algoritma *Divide and Conquer* diambil dari dua kata, yaitu *divide* dan *conquer*. *Divide* yang artinya membagi, digunakan dalam algoritma ini untuk membagi persoalan menjadi beberapa upa-persoalan atau persoalan-persoalan kecil yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. Idealnya, setiap persoalan-persoalan yang telah dibagi ini berukuran sama satu dengan yang lainnya. Kata kedua yaitu *conquer* yang bila diartikan ke bahasa Indonesia adalah menaklukan yang bila dalam algoritma ini diartikan sebagai bagian yang menyelesaikan masalah masing-masing upa-persoalan, baik secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar.

Pemrosesan dari algoritma ini tidak berhenti sampai membagi dan menyelesaikan saja. Terdapat tahapan lebih lanjut yaitu *combine* atau menyatukan dan menggabungkan semua solusi dari masing-masing upa-persoalan sehingga membentuk solusi persoalan semula. Berdasarkan penjelasan-penjelasan tersebut, algoritma ini baik digunakan untuk mengatasi masalah-masalah dengan rekursif alami, selain itu algoritma ini menjadi awal untuk algoritma-algoritma lain yang berbasis *divide and conquer*. Namun, algoritma ini memiliki beberapa kekurangan seperti bisa boros memori jika memang pemanggilan rekursif sangat banyak dan tidak terlalu efisien jika overhead penggabungan (*combine*) terlalu besar.

Pada tugas kecil ini, algoritma *divide and conquer* digunakan untuk kompresi gambar berbasis *quadtree*. Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil dan dalam konteks ini *quadtree* digunakan untuk membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Komponen warna yang dihitung adalah komponen warna RGB, dimana komponen merah, hijau, dan biru dari sebuah gambar akan dihitung tingkat keragamannya. Jika tidak seragam maka akan dibagi menjadi empat bagian dan dihitung lagi, begitu seterusnya sampai sebuah bagian mencapai tingkat keragaman tertentu.

BAB 2

Deskripsi Program

Program kami diimplementasikan menggunakan bahasa Java dengan beberapa library tambahan seperti java.awt untuk pemrosesan gambar. Berikut merupakan beberapa fitur dari program kami:

1. CLI (*Command Line Interface*) based.
2. Pengguna dapat memasukkan file gambar kustom dengan ekstensi jpg, jpeg, ataupun png.
3. Program menyediakan empat metode kompresi gambar yaitu *variance*, *mean absolute deviation* (MAD), *max pixel difference*, dan *entropy*. Pengguna dapat memilih salah satu dari keempat metode tersebut dengan input angka satu sampai empat.
4. Pengguna dapat memasukkan *threshold* atau ambang batas yang menentukan apakah sebuah blok dianggap seragam atau harus dibagi lebih lanjut.
5. Pengguna dapat memasukkan *minimum block size* untuk menentukan ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi.
6. Terakhir, pengguna dapat memasukkan PATH file output secara spesifik, baik itu untuk output gambar, ataupun output gif.
7. Program dapat menghasilkan gambar yang telah dikompresi dengan metode dan spesifikasi yang telah dipilih oleh pengguna.
8. Program juga dapat menghasilkan gif transformasi dari gambar dengan satu blok warna hingga menjadi gambar secara keseluruhan.
9. Program juga akan menampilkan statistik kompresi seperti waktu eksekusi, ukuran gambar sebelum dan sesudah, persentase kompresi, kedalaman pohon, dan banyak simpul pada pohon.

Sebelum menjalankan program, pengguna harus menggunakan sistem operasi yang sudah terinstall java versi terbaru. Berikut adalah cara untuk menjalankan program.

1. Lakukan *clone repository* pada Visual Studio Code.
2. Buka terminal, lalu ketik “java -cp bin Main” untuk menjalankan program.
3. Ikuti arahan program sampai selesai.

Implementasi Program

Garis besar utama pada program ini adalah bagaimana sebuah gambar bisa dikompresi menggunakan metode *quadtree* dengan algoritma dasar *divide and conquer*. Pada program yang telah kami buat, kami membagi program utama menjadi empat buah file utama, yaitu file untuk mengurus *node* atau simpul beserta anak-anaknya, file yang mengurus perhitungan error, file yang mengurus pembentukan *quadtree* itu sendiri, dan file terakhir adalah file main program. Keempat file tersebut saling mendukung satu sama lain untuk melakukan kompresi gambar. Penjelasan rinci terhadap program yang kami buat adalah sebagai berikut:

1. Pada awal program dijalankan, program akan menerima input gambar. Gambar tersebut akan langsung masuk ke fungsi buildTree() pada class Quadtree untuk pemrosesan.
2. Gambar yang sudah masuk ke fungsi buildTree() akan dihitung warna rata-ratanya dengan fungsi avgColor() pada class Node dan akan dibuatkan sebuah node baru.
3. Node baru itu akan dihitung lagi, apakah perlu untuk dibagi atau tidak dengan fungsi divide() di class Quadtree. Definisi perlu disini jika luas dari sebuah gambar masih belum melebihi *minimum block size* dan hasil perhitungan error-nya juga masih lebih besar daripada threshold yang dimasukkan.
4. Perhitungan error akan memanggil fungsi-fungsi pada class ErrorMeasurement sesuai dengan metode yang dipilih saat menjalankan program.
5. Bila memang perlu untuk dibagi, maka fungsi subDivide() pada class Node akan dipanggil untuk membagi node menjadi empat bagian yang disebut *children*.
6. Setiap *children* yang dihasilkan oleh node akan disimpan oleh array yang dimiliki oleh node. Children tersebut akan menjadi node untuk dirinya masing-masing dan akan diproses lagi dengan cara yang sama.
7. Proses akan dilakukan terus sampai fungsi divide() return false dimana pembagian node tidak diperlukan lagi.
8. Jika pembentukan semua node dan children sudah selesai, akan digunakan fungsi reconstructImage() dan fillImage() untuk menyusun kembali gambar. Penyusunan gambar dilakukan secara rekursif, dimana saat sudah menemukan children maka fungsi fillImage() akan mengambil warna dan mengisi bagian tersebut dengan warnanya. Jika bukan children, akan dilakukan rekursif terus menerus hingga menemukan children.

Pendekatan algoritma *divide and conquer* dari program ini dapat dilihat dari node utama yang dibagi menjadi empat children atau *divide* dan empat children tersebut akan diproses tersendiri atau *conquer*. Mereka akan menentukan arahnya sendiri entah itu dibagi lagi atau tidak, jika dibagi maka akan menggunakan pendekatan *divide and conquer* lagi hingga sampai node paling terakhir. Setelah semua proses selesai dilakukan, akan dilakukan proses *combine* dimana semua node tadi akan digabungkan dalam satu gambar yang menciptakan gambar baru sebagai kompresi dari gambar awal.

BAB 3

Source Code

Main.java

```

        System.out.println("Input target kompresi tidak berada dalam rentang yang bisa
dilakukan oleh program ini, sistem akan tetap mencoba.");
    } else {
        System.out.println("Target kompresi diaktifkan! Berusaha menargetkan kompresi ke: "
+ (targetKompresi * 100) + "%");
    }
} else {
    System.out.println("Mode target kompresi dinonaktifkan.");
}

// PATH output
String outputPath = getOutputFilePath(scanner);
File outputFile = new File(outputPath);
if (!prepareOutputFolder(outputFile.getParent())) {
    System.out.println("Folder output tidak tersedia atau tidak dapat dibuat.");
    scanner.close();
    return;
}

// Proses kompresi
long startTime = System.nanoTime();
Quadtree quadtree = new Quadtree(threshold, minBlockSize, metode);
Node root = quadtree.buildTree(image, new Rectangle(0, 0, image.getWidth(),
image.getHeight()));
BufferedImage compressedImage = quadtree.reconstructImage(root, image.getWidth(),
image.getHeight());
long endTime = System.nanoTime();

// Menyimpan gambar hasil kompresi
saveImage(compressedImage, outputFile);

/* Membuat GIF, bisa skip karena heap (jika file terlalu besar, Java tidak cukup, harus
disetting terlebih dahulu) */
boolean skipGIF = false;
if(image.getWidth() * image.getHeight() > 1920*1080){
    System.out.println("\nMaaf! Ukuran gambar terlalu besar untuk membuat GIF proses
kompresi karena memori Java tidak cukup. Kompresi akan tetap dijalankan tanpa pembuatan
GIF.");
    skipGIF = true;
}

if(!skipGIF){
    int maxDepth = quadtree.getDepth(root);
    List<BufferedImage> evolutionFrames = new ArrayList<>();
    for (int level = 1; level <= maxDepth; level++) {
        BufferedImage frame = quadtree.reconstructImageAtLevel(root, level,
image.getWidth(), image.getHeight());
        evolutionFrames.add(frame);
    }

    // Dapatkan lokasi file output hasil GIF secara absolut
    String outputGifPath = getOutputGifPath(scanner);
    try {
        ImageOutputStream evoOutput = new FileImageOutputStream(new File(outputGifPath));
        GifWriter evoGifWriter = new GifWriter(evoOutput, BufferedImage.TYPE_INT_RGB, 50,
true);
        for (BufferedImage frame : evolutionFrames) {
            evoGifWriter.writeToSequence(frame,50);
        }
        evoGifWriter.close();
    } catch (IOException e) {
        e.printStackTrace();
    }
    System.out.println("GIF berhasil disimpan pada path yang dituju.");
}
}

// Output statistik
printStatistics(inputPath, outputFile, quadtree, root, startTime, endTime);

scanner.close();

```

```

}

// Fungsi untuk membaca gambar
private static BufferedImage loadImage(String inputPath) {
    BufferedImage image = null;
    try {
        //System.out.println("Path yang dibaca: " + inputPath);
        System.out.println("File berhasil dibaca.");
        image = ImageIO.read(new File(inputPath));
        if (image == null) {
            System.out.println("Error: Format gambar tidak didukung atau file rusak.");
            return null;
        }
    } catch (IOException e) {
        System.out.println("Gagal membaca gambar! Periksa kembali path file.");
        return null;
    }
    return image;
}

// Fungsi untuk memilih metode
private static ErrorMeasurement.Metode chooseErrorMethod(Scanner scanner) {
    ErrorMeasurement.Metode metode = null;
    while(metode == null) {
        System.out.println("\nList Metode Perhitungan Error");
        System.out.println("1. Variance (0 - 20000)");
        System.out.println("2. MAD (0 - 100)");
        System.out.println("3. MaxPixelDifference (0 - 255)");
        System.out.println("4. Entropy (0.5 - 8)");
        System.out.println("5. SSIM (0 - 1)");
        System.out.print("Pilih metode perhitungan error: ");
        if(scanner.hasNextInt()){
            int metodePilihan = scanner.nextInt();
            scanner.nextLine();
            if (metodePilihan >= 1 && metodePilihan <= 5) {
                metode = ErrorMeasurement.Metode.values()[metodePilihan - 1];
            } else {
                System.out.println("Pilihan metode tidak valid! Silakan coba lagi antara 1-5.");
            }
        } else {
            System.out.println("Input tidak valid. Silakan masukkan angka bulat antara 1-5.");
            scanner.nextLine();
        }
    }
    return metode;
}

// Fungsi untuk mendapatkan threshold
private static double getThreshold(Scanner scanner, ErrorMeasurement.Metode metode) {
    double threshold;
    while (true) {
        System.out.print("Masukkan threshold: ");
        String input = scanner.next().replace(",",".");
        scanner.nextLine();

        try {
            threshold = Double.parseDouble(input);
        } catch (NumberFormatException e) {
            System.out.println("Input tidak valid. Masukkan angka desimal dengan titik atau koma.");
            continue;
        }

        if (metode == ErrorMeasurement.Metode.VARIANCE && (threshold < 0 || threshold > 20000)) {
            System.out.println("Threshold tidak sesuai range untuk metode Variansi.");
        } else if (metode == ErrorMeasurement.Metode.MAD && (threshold < 0 || threshold > 100)) {
            System.out.println("Threshold tidak sesuai range untuk metode MAD.");
        } else if (metode == ErrorMeasurement.Metode.MPD && (threshold < 0 || threshold >

```

```

255)) {
    System.out.println("Threshold tidak sesuai range untuk metode MPD.");
} else if (metode == ErrorMeasurement.Metode.ENTROPY && (threshold < 0.5 || threshold
> 8)) {
    System.out.println("Threshold tidak sesuai range untuk metode Entropi.");
} else if(metode == ErrorMeasurement.Metode.SSIM && (threshold < 0 || threshold > 1)){
    System.out.println("Threshold tidak sesuai range untuk metode SSIM.");
} else {
    break;
}
}
return threshold;
}

// Fungsi untuk mendapatkan ukuran blok minimum
private static int getMinBlockSize(Scanner scanner) {
    System.out.print("Masukkan ukuran blok minimum: ");
    while (!scanner.hasNextInt()) {
        System.out.println("Maaf! Input tidak valid. Silakan masukkan angka bulat.");
        scanner.nextLine();
        System.out.print("Masukkan ukuran blok minimum: ");
    }
    int minBlockSize = scanner.nextInt();
    scanner.nextLine();
    return minBlockSize;
}

// Fungsi untuk memastikan folder output tersedia
private static boolean prepareOutputFolder(String outputFolder) {
    File outputDir = new File(outputFolder);
    if (!outputDir.exists() && !outputDir.mkdirs()) {
        System.out.println("Maaf! Gagal membuat direktori output.");
        return false;
    }
    return true;
}

// Fungsi untuk mendapatkan ekstensi file
private static String getFileExtension(File file) {
    String name = file.getName();
    int dotIndex = name.lastIndexOf('.');
    return (dotIndex == -1) ? "png" : name.substring(dotIndex + 1);
}

// Fungsi untuk menyimpan gambar
private static void saveImage(BufferedImage image, File outputFile) {
    try {
        String extension = getFileExtension(outputFile);
        ImageIO.write(image, extension, outputFile);
        System.out.println("Gambar berhasil disimpan pada path yang dituju.");
    } catch (IOException e) {
        System.out.println("Maaf! Gagal menyimpan gambar! Periksa kembali path file.");
        e.printStackTrace();
    }
}

// Fungsi untuk mencetak statistik hasil kompresi
private static void printStatistics(String inputPath, File outputFile, Quadtree quadtree,
Node root, long startTime,
long endTime) {
    double executionTime = (endTime - startTime) / 1e9;
    long originalSize = new File(inputPath).length();
    long compressedSize = outputFile.length();
    double compressionRatio = ((double) (originalSize - compressedSize) / originalSize) *
100;
    int depth = quadtree.getDepth(root);
    int nodeCount = quadtree.getTotalNodes(root);

    System.out.println("\nStatistik Kompresi Gambar");
    System.out.println("Waktu eksekusi: " + executionTime + " detik");
}

```

```

        System.out.println("Ukuran gambar sebelum: " + originalSize + " bytes");
        System.out.println("Ukuran gambar setelah: " + compressedSize + " bytes");
        System.out.println("Persentase kompresi: " + compressionRatio + "%");
        System.out.println("Kedalaman pohon: " + depth);
        System.out.println("Banyak simpul pada pohon: " + nodeCount);
    }

    /* Fungsi untuk mendapatkan input path dan validasi */
    private static String getInputPath(Scanner scanner) {
        String inputPath;
        BufferedImage testImage = null;
        while (true) {
            System.out.print("Masukkan path absolut gambar (contoh C:\\\\folder\\\\gambar.jpg): ");
            inputPath = scanner.nextLine().trim();
            File file = new File(inputPath);
            if (!file.exists() || !file.isFile()) {
                System.out.println("Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.");
                continue;
            }
            try {
                testImage = ImageIO.read(file);
            } catch (IOException e) {
                System.out.println("Maaf! Terjadi kesalahan saat membaca gambar: " +
e.getMessage());
                continue;
            }
            if (testImage == null) {
                System.out.println("Maaf! Format gambar tidak didukung atau file rusak. Silakan coba lagi.");
                continue;
            }
            break;
        }
        return inputPath;
    }

    /* Fungsi untuk mendapatkan output path dan validasi */
    private static String getOutputFilePath(Scanner scanner) {
        String filePath;
        while (true) {
            System.out.print("\nMasukkan path absolut untuk file hasil kompresi (contoh C:\\\\folder\\\\hasil.jpg): ");
            filePath = scanner.nextLine().trim();
            File file = new File(filePath);
            String parent = file.getParent();
            if (parent == null || parent.isEmpty()) {
                System.out.println("Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.");
            } else {
                break;
            }
        }
        return filePath;
    }

    /* Fungsi untuk mendapatkan output gif dan validasi */
    private static String getOutputGifPath(Scanner scanner) {
        String filePath;
        while (true) {
            System.out.print("\nMasukkan path absolut untuk file GIF hasil (contoh C:\\\\folder\\\\hasil.gif): ");
            filePath = scanner.nextLine().trim();
            if (!filePath.toLowerCase().endsWith(".gif")) {
                System.out.println("Maaf! Path harus diakhiri dengan ekstensi .gif. Silakan coba lagi.");
                continue;
            }
            File file = new File(filePath);
            String parent = file.getParent();

```

```

        if (parent == null || parent.isEmpty()) {
            System.out.println("Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan
hanya nama file.");
        } else {
            break;
        }
    }
    return filePath;
}

/* Fungsi untuk mendapatkan BONUS 1 - Target persentase dan validasi */
private static double getValidatedTarget(Scanner scanner) {
    double target;
    while (true) {
        System.out.print("\nMasukkan target persentase kompresi (0.0-1.0 | 0 untuk
menonaktifkan mode ini): ");
        String input = scanner.nextLine().replace(", ", ".").trim();
        try {
            target = Double.parseDouble(input);
        } catch (NumberFormatException e) {
            System.out.println("Maaf! Input tidak valid. Silakan masukkan angka desimal.");
            continue;
        }
        if (target < 0.0 || target > 1.0) {
            System.out.println("Maaf! Input harus berada di antara 0.0 dan 1.0. Silakan coba
lagi.");
        } else {
            break;
        }
    }
    return target;
}

/* Fungsi untuk BONUS 1, mencari threshold yang sesuai dengan keinginan user */
private static double autoAdjustThreshold(BufferedImage image, long originalSize,
ErrorMeasurement.Metode metode, String format, double target){
    double batasBawah, batasAtas;
    switch(metode){
        case VARIANCE:
            batasBawah = 0; batasAtas = 20000;
            break;
        case MAD:
            batasBawah = 0; batasAtas = 100;
            break;
        case MPD:
            batasBawah = 0; batasAtas = 255;
            break;
        case ENTROPY:
            batasBawah = 0.5; batasAtas = 8;
            break;
        case SSIM:
            batasBawah = 0; batasAtas = 1;
            break;
        default:
            batasBawah = 0; batasAtas = 0;
    }

    double mid = batasBawah;

    for(int i = 0;i < 10;i++){
        mid = (batasBawah + batasAtas) / 2;
        Quadtree qt = new Quadtree(mid, 1, metode);
        Node root = qt.buildTree(image, new
Rectangle(0,0,image.getWidth(),image.getHeight()));
        BufferedImage compressed = qt.reconstructImage(root, image.getWidth(),
image.getHeight());

        byte[] imageBytes = getImageBytes(compressed, format);
        long compressedSize = imageBytes.length;
        double currentCompression = (1 - ((double) compressedSize / (double) originalSize));
    }
}

```

```

//System.out.println("Iterasi " + (i+1) + " - Threshold: " + mid + ", Kompresi Saat
Ini: " + (currentCompression*100) + "%"); // untuk debug
    if(currentCompression < target){
        batasBawah = mid;
    } else {
        batasAtas = mid;
    }
}
return mid;
}

private static byte[] getImageBytes(BufferedImage image, String format){
    ByteArrayOutputStream byteArrayOS = new ByteArrayOutputStream();
    try {
        ImageIO.write(image, format, byteArrayOS);
    } catch (IOException e) {
        e.printStackTrace();
    }
    return byteArrayOS.toByteArray();
}

/* Fungsi untuk BONUS 1, mencari limit bawah dan atas persentase, karena meskipun
threshold 0 atau max, ada batasan sejauh mana dia bisa dikomprimi */
private static double[] getCompressionRange(BufferedImage image, long originalSize,
ErrorMeasurement.Metode metode, String format){
    double batasBawah, batasAtas;
    switch(metode){
        case VARIANCE:
            batasBawah = 0; batasAtas = 20000;
            break;
        case MAD:
            batasBawah = 0; batasAtas = 100;
            break;
        case MPD:
            batasBawah = 0; batasAtas = 255;
            break;
        case ENTROPY:
            batasBawah = 0.5; batasAtas = 8;
            break;
        case SSIM:
            batasBawah = 0; batasAtas = 1;
            break;
        default:
            batasBawah = 0; batasAtas = 0;
    }
    double minCompression = getCompressionForThreshold(image, originalSize, metode, format,
batasBawah);
    double maxCompression = getCompressionForThreshold(image, originalSize, metode, format,
batasAtas);
    return new double[]{minCompression, maxCompression};
}

/* Fungsi untuk BONUS 1, mencari limit bawah dan atas persentase, karena meskipun
threshold 0 atau max, ada batasan sejauh mana dia bisa dikomprimi */
private static double getCompressionForThreshold(BufferedImage image, long originalSize,
ErrorMeasurement.Metode metode, String format, double threshold) {
    Quadtree qt = new Quadtree(threshold, 1, metode);
    Node root = qt.buildTree(image, new Rectangle(0, 0, image.getWidth(),
image.getHeight()));
    BufferedImage compressed = qt.reconstructImage(root, image.getWidth(),
image.getHeight());
    byte[] imageBytes = getImageBytes(compressed, format);
    long compressedSize = imageBytes.length;
    return 1 - ((double) compressedSize / (double) originalSize);
}
}

```

Node.java

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

public class Node {
    private Rectangle area;
    private Color avgColor;
    private Node[] children;

    public Node(Rectangle area, Color avgColor) {
        this.area = area;
        this.avgColor = avgColor;
        this.children = null;
    }

    public Rectangle getArea() {
        return area;
    }

    public Color getColor() {
        return avgColor;
    }

    public Node[] getChildren() {
        return children;
    }

    public boolean isLeaf() {
        return children == null;
    }

    public void subDivide(BufferedImage image, Quadtree quadtree) {
        int halfWidth = area.width / 2;
        int halfHeight = area.height / 2;
        int remainingWidth = area.width - halfWidth;
        int remainingHeight = area.height - halfHeight;

        if (halfWidth < quadtree.minBlockSize || halfHeight < quadtree.minBlockSize) {
            return;
        }

        children = new Node[4];

        Rectangle[] subAreas = {
            new Rectangle(area.x, area.y, halfWidth, halfHeight),
            new Rectangle(area.x + halfWidth, area.y, remainingWidth, halfHeight),
            new Rectangle(area.x, area.y + halfHeight, halfWidth, remainingHeight),
            new Rectangle(area.x + halfWidth, area.y + halfHeight, remainingWidth,
remainingHeight)
        };

        for (int i = 0; i < 4; i++) {
            children[i] = quadtree.buildTree(image, subAreas[i]);
        }
    }

    public static Color avgColor(BufferedImage image, Rectangle area) {
        int red = 0;
        int green = 0;
        int blue = 0;
        int totalPixel = area.width * area.height;

        for (int y = area.y; y < area.y + area.height; y++) {
            for (int x = area.x; x < area.x + area.width; x++) {
                if (x >= image.getWidth() || y >= image.getHeight()) {
                    continue;
                }
                int rgb = image.getRGB(x, y);
                red += (rgb & 0xFF0000) >> 16;
                green += (rgb & 0x00FF00) >> 8;
                blue += (rgb & 0x0000FF);
            }
        }

        return new Color(red / totalPixel, green / totalPixel, blue / totalPixel);
    }
}
```

```

        Color color = new Color(rgb);

        red += color.getRed();
        green += color.getGreen();
        blue += color.getBlue();
    }
}

if (totalPixel == 0) {
    return new Color(0, 0, 0);
}

int avgRed = red / totalPixel;
int avgGreen = green / totalPixel;
int avgBlue = blue / totalPixel;

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

```

Quadtree.java

```

import java.awt.Rectangle;
import java.awt.image.BufferedImage;
import java.awt.Graphics2D;
import java.awt.Color;

public class Quadtree {
    private double threshold;
    public int minBlockSize;
    private ErrorMeasurement.Metode metode;

    public Quadtree(double threshold, int minBlockSize, ErrorMeasurement.Metode metode) {
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.metode = metode;
    }

    public int getDepth(Node node) {
        if (node == null || node.isLeaf()) {
            return 1;
        }
        int maxDepth = 0;
        for (Node child : node.getChildren()) {
            maxDepth = Math.max(maxDepth, getDepth(child));
        }
        return maxDepth + 1;
    }

    public int getTotalNodes(Node node) {
        if (node == null) {
            return 0;
        }
        int count = 1;
        if (!node.isLeaf()) {
            for (Node child : node.getChildren()) {
                count += getTotalNodes(child);
            }
        }
        return count;
    }

    public boolean divide(Node node, BufferedImage image) {
        Rectangle area = node.getArea();

```

```

int luas = area.width * area.height;

if (luas < minBlockSize) {
    return false;
}

double err = hitungErr(image, area);
return err > threshold;
}

private double hitungErr(BufferedImage image, Rectangle area) {
    ErrorMeasurement em = new ErrorMeasurement();

    switch (metode) {
        case VARIANCE:
            return em.variance(image, area);
        case MAD:
            return em.mad(image, area);
        case MPD:
            return em.maxPixelDifference(image, area);
        case ENTROPY:
            return em.entropy(image, area);
        case SSIM:
            double error = 1 - em.ssim(image, area);
            // System.out.println("Error = " + error);
            return error;
        default:
            throw new IllegalArgumentException("Metode tidak ada!");
    }
}

public Node buildTree(BufferedImage image, Rectangle area) {
    Color avgColor = Node.avgColor(image, area);
    Node node = new Node(area, avgColor);

    if (divide(node, image)) {
        node.subDivide(image, this);
    }

    return node;
}

public BufferedImage reconstructImage(Node root, int width, int height) {
    BufferedImage newImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
    Graphics2D graphics = newImage.createGraphics();

    fillImage(graphics, root);

    graphics.dispose();
    return newImage;
}

private void fillImage(Graphics2D graphics, Node node) {
    if (node == null) {
        return;
    }

    if (node.isLeaf()) {
        graphics.setColor(node.getColor());
        graphics.fillRect(node.getArea().x, node.getArea().y, node.getArea().width,
        node.getArea().height);
    } else {
        for (Node child : node.getChildren()) {
            fillImage(graphics, child);
        }
    }
}

public BufferedImage reconstructImageAtLevel(Node root, int targetLevel, int width, int
height) {

```

```

        BufferedImage newImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = newImage.createGraphics();
        fillImageAtLevel(g, root, 1, targetLevel);
        g.dispose();
        return newImage;
    }

    private void fillImageAtLevel(Graphics2D g, Node node, int currentLevel, int targetLevel)
    {
        if (node == null)
            return;
        if (node.isLeaf() || currentLevel == targetLevel) {
            g.setColor(node.getColor());
            Rectangle r = node.getArea();
            g.fillRect(r.x, r.y, r.width, r.height);
        } else {
            for (Node child : node.getChildren()) {
                fillImageAtLevel(g, child, currentLevel + 1, targetLevel);
            }
        }
    }
}

```

ErrorMeasurement.java

```

import java.awt.*;
import java.awt.image.BufferedImage;

public class ErrorMeasurement {
    public enum Metode {
        VARIANCE, MAD, MPD, ENTROPY, SSIM;
    }

    public double variance(BufferedImage image, Rectangle area) {
        int jumlahPiksel = area.width * area.height;
        if (jumlahPiksel == 0) {
            return 0;
        }

        // Rata-rata tiap warna
        Color rataWarna = Node.avgColor(image, area);
        double meanRed = rataWarna.getRed();
        double meanGreen = rataWarna.getGreen();
        double meanBlue = rataWarna.getBlue();

        // Menghitung variansi tiap warna
        double variansiRed = 0;
        double variansiGreen = 0;
        double variansiBlue = 0;

        for (int y = area.y; y < area.y + area.height; y++) {
            for (int x = area.x; x < area.x + area.width; x++) {
                if (x >= image.getWidth() || y >= image.getHeight()) {
                    continue;
                }
                Color rgb = new Color(image.getRGB(x, y));
                variansiRed += Math.pow(rgb.getRed() - meanRed, 2);
                variansiGreen += Math.pow(rgb.getGreen() - meanGreen, 2);
                variansiBlue += Math.pow(rgb.getBlue() - meanBlue, 2);
            }
        }

        variansiRed /= jumlahPiksel;
        variansiGreen /= jumlahPiksel;
        variansiBlue /= jumlahPiksel;
    }
}

```

```

variansiBlue /= jumlahPiksel;

    return ((variansiRed + variansiGreen + variansiBlue) / 3);
}

public double mad(BufferedImage image, Rectangle area) {
    int jumlahPiksel = area.width * area.height;
    if (jumlahPiksel == 0) {
        return 0;
    }

    // Rata-rata tiap warna
    Color rataWarna = Node.avgColor(image, area);
    double meanRed = rataWarna.getRed();
    double meanGreen = rataWarna.getGreen();
    double meanBlue = rataWarna.getBlue();

    // Menghitung variansi tiap warna
    double madRed = 0;
    double madGreen = 0;
    double madBlue = 0;

    for (int y = area.y; y < area.y + area.height; y++) {
        for (int x = area.x; x < area.x + area.width; x++) {
            if (x >= image.getWidth() || y >= image.getHeight()) {
                continue;
            }
            Color rgb = new Color(image.getRGB(x, y));
            madRed += Math.abs(rgb.getRed() - meanRed);
            madGreen += Math.abs(rgb.getGreen() - meanGreen);
            madBlue += Math.abs(rgb.getBlue() - meanBlue);
        }
    }

    madRed /= jumlahPiksel;
    madGreen /= jumlahPiksel;
    madBlue /= jumlahPiksel;

    return ((madRed + madGreen + madBlue) / 3);
}

public double maxPixelDifference(BufferedImage image, Rectangle area) {
    int minRed = 255, maxRed = 0;
    int minGreen = 255, maxGreen = 0;
    int minBlue = 255, maxBlue = 0;

    for (int y = area.y; y < area.y + area.height; y++) {
        for (int x = area.x; x < area.x + area.width; x++) {
            if (x >= image.getWidth() || y >= image.getHeight()) {
                continue;
            }

            Color rgb = new Color(image.getRGB(x, y));

            // Update nilai minimum
            minRed = Math.min(minRed, rgb.getRed());
            minGreen = Math.min(minGreen, rgb.getGreen());
            minBlue = Math.min(minBlue, rgb.getBlue());

            // Update nilai maksimum
            maxRed = Math.max(maxRed, rgb.getRed());
            maxGreen = Math.max(maxGreen, rgb.getGreen());
            maxBlue = Math.max(maxBlue, rgb.getBlue());
        }
    }

    int deltaRed = maxRed - minRed;
    int deltaGreen = maxGreen - minGreen;
    int deltaBlue = maxBlue - minBlue;
}

```

```

        return ((deltaRed + deltaGreen + deltaBlue) / 3);
    }

    public double entropy(BufferedImage image, Rectangle area) {
        int jumlahPiksel = area.width * area.height;
        if (jumlahPiksel == 0) {
            return 0;
        }

        // Menghitung kemunculan setiap komponen warna
        int[] histogramRed = new int[256];
        int[] histogramGreen = new int[256];
        int[] histogramBlue = new int[256];

        for (int y = area.y; y < area.y + area.height; y++) {
            for (int x = area.x; x < area.x + area.width; x++) {
                if (x >= image.getWidth() || y >= image.getHeight()) {
                    continue;
                }

                Color pixel = new Color(image.getRGB(x, y));
                histogramRed[pixel.getRed()]++;
                histogramGreen[pixel.getGreen()]++;
                histogramBlue[pixel.getBlue()]++;
            }
        }

        // Menghitung probabilitasnya
        double[] probRed = new double[256];
        double[] probGreen = new double[256];
        double[] probBlue = new double[256];

        for (int i = 0; i < 256; i++) {
            probRed[i] = (double) histogramRed[i] / jumlahPiksel;
            probGreen[i] = (double) histogramGreen[i] / jumlahPiksel;
            probBlue[i] = (double) histogramBlue[i] / jumlahPiksel;
        }

        // Menghitung entropi
        double entropyRed = 0;
        double entropyGreen = 0;
        double entropyBlue = 0;

        for (int i = 0; i < 256; i++) {
            if (probRed[i] > 0) {
                entropyRed -= probRed[i] * (Math.log(probRed[i]) / Math.log(2));
            }
            if (probGreen[i] > 0) {
                entropyGreen -= probGreen[i] * (Math.log(probGreen[i]) / Math.log(2));
            }
            if (probBlue[i] > 0) {
                entropyBlue -= probBlue[i] * (Math.log(probBlue[i]) / Math.log(2));
            }
        }

        return ((entropyRed + entropyGreen + entropyBlue) / 3);
    }

    /* SSIM hanya membandingkan representasi dari area yang dianggap homogen, karena belum
     punya gambar terkompresinya */
    public double ssim(BufferedImage image, Rectangle area) {
        int jumlahPiksel = area.width * area.height;
        if (jumlahPiksel == 0) {
            return 0;
        }

        // Inisiasi setiap channel RGB
        double sumR = 0, sumG = 0, sumB = 0;
        for (int y = area.y; y < area.y + area.height; y++) {
            for (int x = area.x; x < area.x + area.width; x++) {

```

```

        if (x >= image.getWidth() || y >= image.getHeight()) {
            continue;
        }
        Color c = new Color(image.getRGB(x, y));
        sumR += c.getRed();
        sumG += c.getGreen();
        sumB += c.getBlue();
    }
}

// Rata-rata untuk masing-masing channel warna
double muR = sumR / jumlahPiksel;
double muG = sumG / jumlahPiksel;
double muB = sumB / jumlahPiksel;

// Menghitung variansi (kontras) masing-masing channel
double varR = 0, varG = 0, varB = 0;
for (int y = area.y; y < area.y + area.height; y++) {
    for (int x = area.x; x < area.x + area.width; x++) {
        if (x >= image.getWidth() || y >= image.getHeight()) {
            continue;
        }
        Color c = new Color(image.getRGB(x, y));
        varR += Math.pow(c.getRed() - muR, 2);
        varG += Math.pow(c.getGreen() - muG, 2);
        varB += Math.pow(c.getBlue() - muB, 2);
    }
}
varR /= jumlahPiksel;
varG /= jumlahPiksel;
varB /= jumlahPiksel;

// Konstanta untuk 8-bit per kanal
double C2 = Math.pow(0.03 * 255, 2);

// Karena blok referensi (hasil kompresi) adalah blok homogen dengan nilai rata-rata
// perhitungan SSIM per kanal hanya didasarkan pada kontrasnya
double ssimR = C2 / (varR + C2);
double ssimG = C2 / (varG + C2);
double ssimB = C2 / (varB + C2);

double ssimRGB = 0.3 * ssimR + 0.3 * ssimG + 0.4 * ssimB;
return ssimRGB;
}
}

```

GifWriter.java

```

import javax.imageio.*;
import javax.imageio.metadata.*;
import javax.imageio.stream.ImageOutputStream;
import java.awt.image.*;
import java.io.*;

public class GifWriter {
    private ImageWriter writer;
    private ImageWriteParam param;
    private IIOMetadata metadata;
    private ImageOutputStream stream;

    public GifWriter(ImageOutputStream stream, int imageType, int

```

```

delayTimeCS, boolean loop) throws IOException {
    this.stream = stream;
    writer = ImageIO.getImageWritersByFormatName("gif").next();
    param = writer.getDefaultWriteParam();
    ImageTypeSpecifier typeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
    metadata = writer.getDefaultImageMetadata(typeSpecifier,
param);

    configureRootMetadata(delayTimeCS, loop);
    writer.setOutput(stream);
    writer.prepareWriteSequence(null);
}

private void configureRootMetadata(int delayTimeCS, boolean
loop) throws IIOInvalidTreeException {
    String metaFormat = metadata.getNativeMetadataFormatName();
    IIOMetadataNode root = new IIOMetadataNode(metaFormat);

    IIOMetadataNode gce = new
IIOMetadataNode("GraphicControlExtension");
    gce.setAttribute("disposalMethod", "none");
    gce.setAttribute("userInputFlag", "FALSE");
    gce.setAttribute("transparentColorFlag", "FALSE");
    gce.setAttribute("delayTime",
Integer.toString(delayTimeCS));
    gce.setAttribute("transparentColorIndex", "0");
    root.appendChild(gce);

    IIOMetadataNode appExtensions = new
IIOMetadataNode("ApplicationExtensions");
    IIOMetadataNode appNode = new
IIOMetadataNode("ApplicationExtension");

    appNode.setAttribute("applicationID", "NETSCAPE");
    appNode.setAttribute("authenticationCode", "2.0");

    byte[] loopBytes = new byte[]{1, (byte) (loop ? 0 : 1), 0};
    appNode.setUserObject(loopBytes);
    appExtensions.appendChild(appNode);
    root.appendChild(appExtensions);

    metadata.setFromTree(metaFormat, root);
}

public void writeToSequence(RenderedImage img, int delayTimeCS)
throws IOException {
    IIOMetadata frameMetadata = writer.getDefaultImageMetadata(
        ImageTypeSpecifier.createFromRenderedImage(img),
param);
}

```

```
        String metaFormat =
frameMetadata.getNativeMetadataFormatName();
        IIOMetadataNode root = new IIOMetadataNode(metaFormat);

        IIOMetadataNode gce = new
IIOMetadataNode("GraphicControlExtension");
        gce.setAttribute("disposalMethod", "none");
        gce.setAttribute("userInputFlag", "FALSE");
        gce.setAttribute("transparentColorFlag", "FALSE");
        gce.setAttribute("delayTime",
Integer.toString(delayTimeCS));
        gce.setAttribute("transparentColorIndex", "0");
        root.appendChild(gce);

        frameMetadata.mergeTree(metaFormat, root);
        writer.writeToSequence(new IIIOImage(img, null,
frameMetadata), param);
    }

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

BAB 4

Pengujian

Pada setiap pengujian, akan diberikan tampilan gambar berupa masukan pengguna (input), gambar asli, gambar hasil kompresi, dan GIF proses pembentukan Quadtree (jika ada).

Pengujian 1 - Kompresi Gambar Normal Menggunakan Metode Pengukuran Error Variance (Threshold 250, Minimum Block Size 4, No Target Compression)

```
PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\tucil2_13523018_13523113> java -cp bin Main

Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\tucil2_13523018_13523113\test\input\2600947.jpg
File berhasil dibaca.

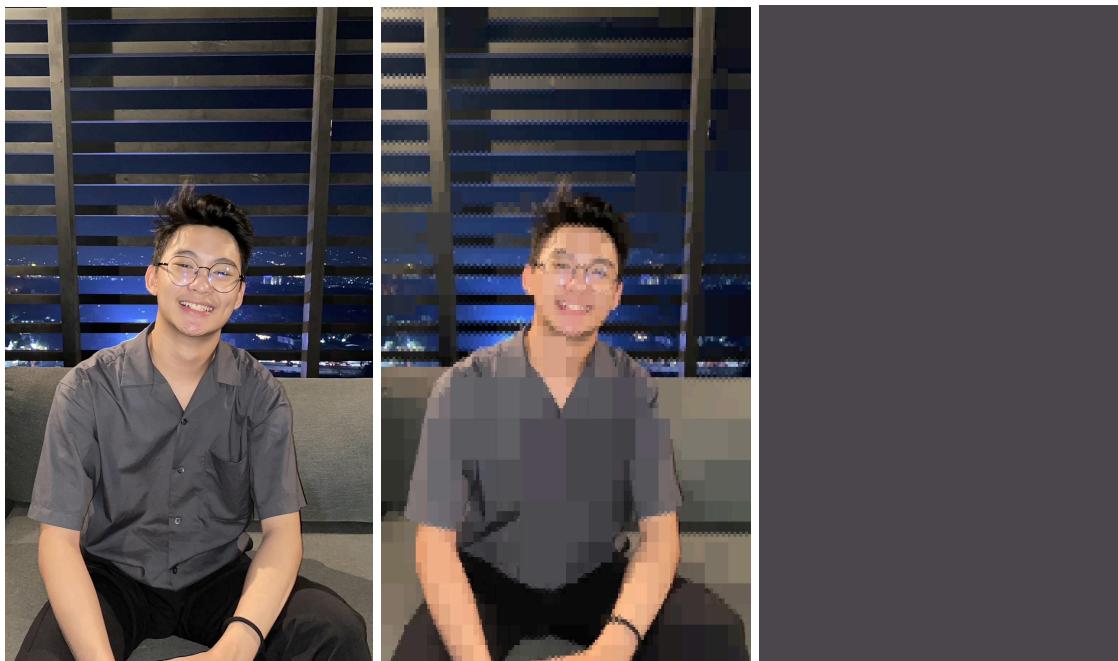
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 100)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 1
Masukkan threshold: 250
Masukkan ukuran blok minimum: 4

Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0
Mode target kompresi dinonaktifkan.

Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\tucil2_13523018_13523113\test\input\2600947_hasil.jpg
Gambar berhasil disimpan pada path yang dituju.

Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\tucil2_13523018_13523113\test\input\2600947_hasil.gif
GIF berhasil disimpan pada path yang dituju.

Statistik Kompresi Gambar
Waktu eksekusi: 1.0159798 detik
Ukuran gambar sebelum: 334152 bytes
Ukuran gambar setelah: 121739 bytes
Persentase kompresi: 63.56777538365775%
Kedalaman pohon: 9
Banyak simpul pada pohon: 13685
◆ PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\tucil2_13523018_13523113>
```



Pengujian 2 - Kompresi Gambar Normal Menggunakan Metode Pengukuran Error MAD (Threshold 10, Minimum Block Size 4, No Target Compression)

```
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\porsche718.jpg  
File berhasil dibaca.  
  
List Metode Perhitungan Error  
1. Variance (0 - 20000)  
2. MAD (0 - 100)  
3. MaxPixelDifference (0 - 255)  
4. Entropy (0.5 - 8)  
5. SSIM (0 - 1)  
Pilih metode perhitungan error: 2  
Masukkan threshold: 10  
Masukkan ukuran blok minimum: 4  
  
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0  
Mode target kompresi dinonaktifkan.  
  
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil.jpg  
Gambar berhasil disimpan pada path yang dituju.  
  
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil.gif  
GIF berhasil disimpan pada path yang dituju.  
  
Statistik Kompresi Gambar  
Waktu eksekusi: 1.0401329 detik  
Ukuran gambar sebelum: 276965 bytes  
Ukuran gambar setelah: 81181 bytes  
Persentase kompresi: 70.68907623706967%  
Kedalaman pohon: 9  
Banyak simpul pada pohon: 7717  
PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113>
```





Pengujian 3 - Kompresi Gambar Normal Menggunakan Metode Pengukuran Error Max Pixel Difference (MPD) (Threshold 100, Minimum Block Size 2, No Target Compression)

```
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\wawancara.jpg
File berhasil dibaca.

List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 3
Masukkan threshold: 100
Masukkan ukuran blok minimum: 2

Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0
Mode target kompresi dinonaktifkan.

Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\wawancara_hasil.jpg
Gambar berhasil disimpan pada path yang dituju.

Maaf! Ukuran gambar terlalu besar untuk membuat GIF proses kompresi karena memori Java tidak cukup. Kompresi akan tetap dijalankan tanpa pembuatan GIF.

Statistik Kompresi Gambar
Waktu eksekusi: 4.447425 detik
Ukuran gambar sebelum: 1866814 bytes
Ukuran gambar setelah: 435781 bytes
Persentase kompresi: 76.6564317602075%
Kedalaman pohon: 11
Banyak simpul pada pohon: 32741
```



Pengujian 4 - Kompresi Gambar Normal Menggunakan Metode Pengukuran Error Entropy (Threshold 1, Minimum Block Size 4, No Target Compression)

```

Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\timuraje.jpg
File berhasil dibaca.

List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 4
Masukkan threshold: 1
Masukkan ukuran blok minimum: 4

Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0
Mode target kompresi dinonaktifkan.

Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\timuraje_hasil.jpg
Gambar berhasil disimpan pada path yang dituju.

Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\timuraje_hasil.gif
GIF berhasil disimpan pada path yang dituju.

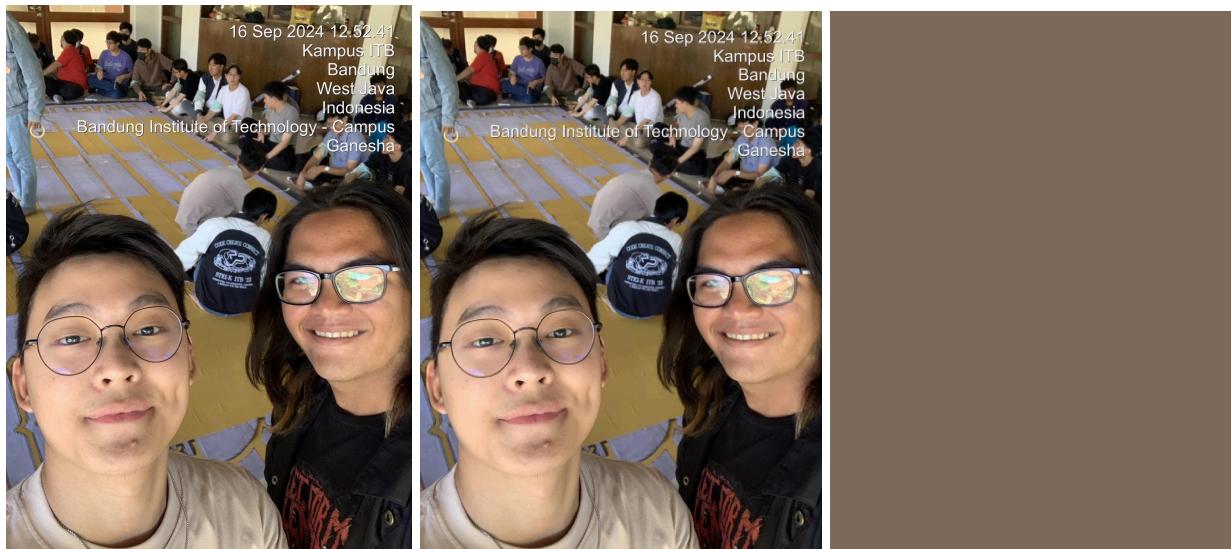
Statistik Kompresi Gambar
Waktu eksekusi: 1.5422492 detik
Ukuran gambar sebelum: 390461 bytes
Ukuran gambar setelah: 147450 bytes
Persentase kompresi: 62.236945559223585%
Kedalaman pohon: 9
Banyak simpul pada pohon: 87/189
PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113> █

```



Pengujian 5 - Kompresi Gambar Normal Menggunakan Metode Pengukuran Error SSIM (Threshold 0.1, Minimum Block Size 2, No Target Compression)

```
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\japrulnih.jpg  
File berhasil dibaca.  
  
List Metode Perhitungan Error  
1. Variance (0 - 20000)  
2. MAD (0 - 255)  
3. MaxPixelDifference (0 - 255)  
4. Entropy (0.5 - 8)  
5. SSIM (0 - 1)  
Pilih metode perhitungan error: 5  
Masukkan threshold: 0,1  
Masukkan ukuran blok minimum: 2  
  
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0  
Mode target kompresi dinonaktifkan.  
  
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\japrulnih_hasil.jpg  
Gambar berhasil disimpan pada path yang dituju.  
  
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\japrulnih_hasil.gif  
GIF berhasil disimpan pada path yang dituju.  
  
statistik Kompresi Gambar  
Waktu eksekusi: 0.819202 detik  
Ukuran gambar sebelum: 461680 bytes  
Ukuran gambar setelah: 210006 bytes  
Percentase kompresi: 54.51264945416738%  
Kedalaman pohon: 10  
Banyak simpul pada pohon: 284385
```



Pengujian 6 - Kompresi Gambar Menggunakan SSIM dan Target Persentase 0.6 atau 60% pada File Besar

```
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\2720825.jpg
File berhasil dibaca.

List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 5
Masukkan threshold: 0
Masukkan ukuran blok minimum: 1

Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0,6
Rentang target kompresi untuk metode SSIM adalah: 57,85% - 95,65%
Target kompresi diaktifkan! Berusaha menargetkan kompresi ke: 60,0%

Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\2720825.hasil.jpg
Gambar berhasil disimpan pada path yang dituju.

Maaf! Ukuran gambar terlalu besar untuk membuat GIF proses kompresi karena memori Java tidak cukup. Kompresi akan tetap dijalankan tanpa pembuatan GIF.

Statistik Kompresi Gambar
Waktu eksekusi: 5.5711568 detik
Ukuran gambar sebelum: 4397236 bytes
Ukuran gambar setelah: 1758936 bytes
Persentase kompresi: 59.99905395116387%
Kedalaman pohon: 13
Banyak simpul pada pohon: 3810533
PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113>
```



Pengujian 7 - Pengujian Bonus Kompresi Gambar dengan Target Persentase 50, Metode SSIM, GIF

```
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\kawishmif.jpg  
File berhasil dibaca.
```

```
List Metode Perhitungan Error
```

1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)

```
Pilih metode perhitungan error: 5
```

```
Masukkan threshold: 0,1
```

```
Masukkan ukuran blok minimum: 99999
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0,5
```

```
Rentang target kompresi untuk metode SSIM adalah: 44,15% - 92,29%
```

```
Target kompresi diaktifkan! Berusaha menargetkan kompresi ke: 50.0%
```

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\kawishmif_hasil.jpg
```

```
Gambar berhasil disimpan pada path yang dituju.
```

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\kawishmif_hasil.gif
```

```
GIF berhasil disimpan pada path yang dituju.
```

```
Statistik Kompresi Gambar
```

```
Waktu eksekusi: 0.4838504 detik
```

```
Ukuran gambar sebelum: 340107 bytes
```

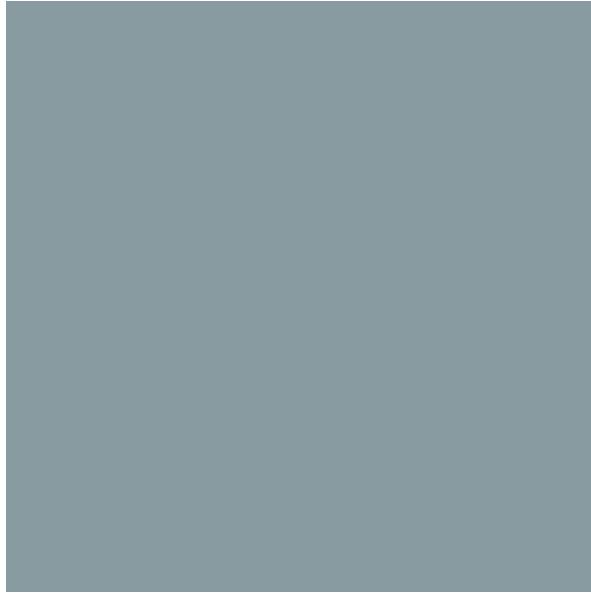
```
Ukuran gambar setelah: 170184 bytes
```

```
Persentase kompresi: 49.96162972241089%
```

```
Kedalaman pohon: 12
```

```
Banyak simpul pada pohon: 248273
```





Pengujian 8 - Kompresi Gambar menggunakan SSIM pada Gambar Sangat Kecil (Threshold 0, Minimum Block Size 1, Target Persentase OFF)

```
Masukkan path absolut gambar (contoh c:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\tesbnw.jpg
File berhasil dibaca.

List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 5
Masukkan threshold: 0
Masukkan ukuran blok minimum: 1

Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0
Mode target kompresi dinonaktifkan.

Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\tesbnw_hasil.jpg
Gambar berhasil disimpan pada path yang dituju.

Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\tesbnw_hasil.gif
GIF berhasil disimpan pada path yang dituju.

Statistik Kompresi Gambar
Waktu eksekusi: 0.2409918 detik
Ukuran gambar sebelum: 35622 bytes
Ukuran gambar setelah: 36199 bytes
Persentase kompresi: -1.6197855257986635%
Kedalaman pohon: 11
Banyak simpul pada pohon: 234385
```



Pengujian 9 - Kompresi Gambar Size Besar dengan Metode SSIM (Threshold 0,1, Minimum Block Size 1, Target Persentase OFF)

```
Masukkan path absolut gambar (contoh c:\folder\gambar.jpg): c:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\b.png  
File berhasil dibaca.
```

```
List Metode Perhitungan Error
```

1. Variance (θ - 20000)
2. MAD (θ - 255)
3. MaxPixelDifference (θ - 255)
4. Entropy (0.5 - 8)
5. SSIM (θ - 1)

```
Pilih metode perhitungan error: 5
```

```
Masukkan threshold: 0,1
```

```
Masukkan ukuran blok minimum: 1
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0  
Mode target kompresi dinonaktifkan.
```

```
Masukkan path absolut untuk file hasil kompresi (contoh c:\folder\hasil.jpg): c:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\b_hasil.png  
Gambar berhasil disimpan pada path yang dituju.
```

```
Maaf! Ukuran gambar terlalu besar untuk membuat GIF proses kompresi karena memori Java tidak cukup. Kompresi akan tetap dijalankan tanpa pembuatan GIF.
```

```
Statistik Kompresi Gambar
```

```
Waktu eksekusi: 7.4305888 detik
```

```
Ukuran gambar sebelum: 11122599 bytes
```

```
Ukuran gambar setelah: 4009901 bytes
```

```
Persentase kompresi: 63.94816535236054%
```

```
Kedalaman pohon: 13
```

```
Banyak simpul pada pohon: 2509661
```



WEB DEVELOPMENT

Stitch Loop:
Where Every Thread Tells a New Story

SYARAT DAN KETENTUAN

- Mahasiswa aktif jenjang **D3/D4/S1** dari perguruan tinggi di Seluruh Indonesia
- Setiap tim terdiri dari **maksimal 3 orang** dari universitas yang sama
- Tim yang lolos ke tahap final, akan **melakukan presentasi** di Auditorium Fakultas Teknologi Industri, Kampus St. Bonaventura, Universitas Atma Jaya Yogyakarta

PENDAFTARAN
Rp 50.000,- / Tim

PRIZEPOOL
Rp 10.500.000,-
TOKEN KELAS DICODING
+ SERTIFIKAT NASIONAL + TROPHY

08112639082
Vela
087829407249
Aldo

COMPETITION

PENDAFTARAN
17 FEB - 08 MAR

PENGUMPULAN
08 MAR

PENJURIAN
12 MAR - 20 MAR

PRESENTASI FINALIS TECHNICAL MEETING PENGUMUMAN
11 APR 22 MAR 22 MAR

SPONSORED BY :



[bit.ly/
Pendaftaran-WDC2025](http://bit.ly/Pendaftaran-WDC2025)



[bit.ly/
RulebookWDC2025](http://bit.ly/RulebookWDC2025)

MEDIA PARTNER





WEB DEVELOPMENT

Stitch Loop:
Where Every Thread Tells a New Story

SYARAT DAN KETENTUAN

- Mahasiswa aktif jenjang **D3/D4/S1** dari perguruan tinggi di Seluruh Indonesia
- Setiap tim terdiri dari **maksimal 3 orang** dari universitas yang sama
- Tim yang lolos ke tahap final, akan melakukan **presentasi** di Auditorium Fakultas Teknologi Industri, Kampus St. Bonaventura, Universitas Atma Jaya Yogyakarta

PENDAFTARAN
Rp 50.000,- / Tim

PRIZEPOOL
Rp 10.500.000,-
TOKEN KELAS DICODING
+ SERTIFIKAT NASIONAL + TROPHY

08112639082
Vela
087829407249
Aldo

COMPETITION

PENDAFTARAN
17 FEB - 08 MAR

PENGUMPULAN
08 MAR

PENJURIAN
12 MAR - 20 MAR

PRESENTASI FINALIS TECHNICAL MEETING PENGUMUMAN
11 APR

SPONSORED BY :



[bit.ly/
Pendaftaran-WDC2025](http://bit.ly/Pendaftaran-WDC2025)



[bit.ly/
RulebookWDC2025](http://bit.ly/RulebookWDC2025)

MEDIA PARTNER



Pengujian 10 - Pengujian Validasi Input

Input Path Absolut Image Input (harus dari root, sampai .jpg atau .png dan sejenisnya)

```
PS C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113> java -cp bin Main
[REDACTED]
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): aaaaaa.jpg
Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\porsche718
Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.
Masukkan path absolut gambar (contoh C:\folder\gambar.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\porsche718.jpg
File berhasil dibaca.

List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 1
```

Input Pemilihan Metode Perhitungan Error

```
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: -1
Pilihan metode tidak valid! Silakan coba lagi antara 1-5.
```

```
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: a
Input tidak valid. Silakan masukkan angka bulat antara 1-5.
```

```
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 6
Pilihan metode tidak valid! Silakan coba lagi antara 1-5.
```

Input Pemilihan Threshold

```
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 1
Masukkan threshold: -1
Threshold tidak sesuai range untuk metode Variansi.
Masukkan threshold: 99999
Threshold tidak sesuai range untuk metode Variansi.
Masukkan threshold: aa
Input tidak valid. Masukkan angka desimal dengan titik atau koma.
Masukkan threshold: 500
Masukkan ukuran blok minimum (> 0): 
```

Input Pemilihan Minimum Block Size

```
List Metode Perhitungan Error
1. Variance (0 - 20000)
2. MAD (0 - 255)
3. MaxPixelDifference (0 - 255)
4. Entropy (0.5 - 8)
5. SSIM (0 - 1)
Pilih metode perhitungan error: 1
Masukkan threshold: 999
Masukkan ukuran blok minimum (> 0): 0
Ukuran blok minimum harus lebih besar dari 0.
Masukkan ukuran blok minimum (> 0): 0,2
Input tidak valid. Silakan masukkan angka bulat.
Masukkan ukuran blok minimum (> 0): -1
Ukuran blok minimum harus lebih besar dari 0.
Masukkan ukuran blok minimum (> 0): 4
```

Input Target Persentase Kompresi

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): aa  
Maaf! Input tidak valid. Silakan masukkan angka desimal.
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): -1  
Maaf! Input harus berada di antara 0.0 dan 1.0. Silakan coba lagi.
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 1,1  
Maaf! Input harus berada di antara 0.0 dan 1.0. Silakan coba lagi.
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0  
Mode target kompresi dinonaktifkan.
```

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): []
```

Input Path Absolut Gambar Hasil Kompresi

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil2  
Maaf! Ekstensi file tidak valid. Pastikan ekstensi file adalah .jpg, .jpeg, atau .png.
```

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): aaaa  
Maaf! Ekstensi file tidak valid. Pastikan ekstensi file adalah .jpg, .jpeg, atau .png.
```

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): hasil.jpg  
Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.
```

```
Masukkan path absolut untuk file hasil kompresi (contoh C:\folder\hasil.jpg): C:\Users\kefas\Documents\VSCode\ITB135\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil2.jpg  
Gambar berhasil disimpan pada path yang dituju.
```

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): []
```

Input Path Absolut GIF

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): aaa  
Maaf! Path harus diakhiri dengan ekstensi .gif. Silakan coba lagi.
```

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil2.jpg  
Maaf! Path harus diakhiri dengan ekstensi .gif. Silakan coba lagi.
```

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): hasil.gif  
Maaf! Path tidak valid. Pastikan menyertakan direktori, bukan hanya nama file.
```

```
Masukkan path absolut untuk file GIF hasil (contoh C:\folder\hasil.gif): C:\Users\kefas\Documents\VSCode\ITB135\STIMA\Tucil2\Tucil2_13523018_13523113\test\output\porsche718_hasil2.gif  
GIF berhasil disimpan pada path yang dituju.
```

Statistik Komprimasi Gambar

Waktu eksekusi: 1.067638 detik

Ukuran gambar sebelum: 276965 bytes

Ukuran gambar setelah: 88714 bytes

Persentase komprimasi: 67.96923799035979%

Kedalaman pohon: 12

Banyak simpul pada pohon: 58717

BAB 5

Analisis

Analisis Hasil

Berdasarkan beberapa pengujian yang telah kami lakukan, kami mendapatkan hasil sebagai berikut. Pertama, pada beberapa pengujian sudah mengimplementasikan kompresi gambar dengan baik, dilihat dari hasil gambar yang berstruktur kotak dengan warna rata-rata menjadi warna utama serta ukuran file yang berkurang. Berdasarkan hal tersebut, kami menemukan adanya “anomali” dalam dua pengujian kami yang menggunakan metode SSIM, terkhusus pada pengujian 8 dan pengujian 9. Pada pengujian 8, hasil dari kompresi gambar didapatkan dengan ukuran yang lebih besar dari gambar awal. Hal tersebut dikarenakan gambar awal sudah memiliki ukuran yang sangat kecil. Oleh karena itu, setelah dilakukan pemrosesan gambar yang terdiri dari banyak proses seperti menghitung ukuran blok, pembagian gambar, dll, dihasilkan gambar dengan ukuran yang lebih besar dari awal.

Berbeda dengan pengujian 9, dimana pada pengujian ini kami menggunakan gambar dengan ukuran yang cukup besar yaitu 11mb. Setelah dilakukan proses kompresi, didapatkan gambar dengan ukuran 4mb saja dan total kompresi adalah 63%. Uniknya, gambar yang dihasilkan masih sangat mirip dengan gambar awal, dengan detail yang sangat bagus juga. Setelah kami analisis, hal tersebut dikarenakan gambar dengan ukuran besar biasanya memiliki area yang lebar dan warna yang relatif seragam pada area yang lebar tersebut. Terlebih lagi kami menggunakan metode SSIM yang menghargai struktur visual, bukan hanya piksel mentah saja. Jadi meskipun piksel tidak 100% sama, pola dan kontrasnya tetap mirip. Kombinasi *threshold* dan *minimum blok size* yang kami gunakan juga turut mendukung hal tersebut dan dihasilkan gambar dengan ketajaman tinggi namun ukuran yang jauh lebih kecil dari gambar awal.

Kompleksitas Algoritma

Kami telah melakukan banyak testing dan pengujian. Berdasarkan hal-hal tersebut, kami menganalisis beberapa hal dari program kami. Sesuai yang sudah disebutkan sebelumnya, program kami dibuat dengan pendekatan *Divide and Conquer* dimana gambar yang dimasukan akan dibagi menjadi blok-blok *children* yang lebih kecil secara rekursif. Hal ini akan terus dilakukan sampai hasil perhitungan error blok sudah dibawah *threshold* yang ditentukan atau ukuran blok telah mencapai batas minimum.

Perhitungan yang kami lakukan untuk kompleksitas algoritma ini dimulai dengan menganggap ukuran gambar adalah n piksel, dimana n ini dihitung dari luas gambar sendiri yaitu panjang dikali lebar gambar. Quadtree akan memproses gambar ini dengan membagi gambar menjadi empat bagian. Saat terjadi pembagian ini, jumlah piksel yang diproses pada tiap level

tetap tetap yaitu $O(n)$ dikarenakan walaupun blok makin kecil, jumlah node bertambah yang membuat jumlah piksel akan tetap.

Kemudian, terdapat perhitungan kompleksitas algoritma untuk pembagian blok gambar. Pembagian blok gambar hanya bisa dilakukan dengan pembagian dua, misal ukuran awal gambar adalah 64×64 maka gambar akan dibagi menjadi empat dengan masing-masing gambar berukuran 32×32 . Proses ini akan terus dilakukan hingga mencapai ukuran blok minimum. Pembagian akan dilakukan sebanyak $2 \log (\text{sisi})$ kali. Perhitungan kompleksitas dapat dihitung dari sini, yaitu dari kedalaman pohon. Kompleksitas dihitung dari kedalaman maksimal pohon yaitu $2 \log (\text{root}(n))$ atau sama dengan $(\frac{1}{2}) 2 \log (n)$ dan bila dimasukkan ke dalam notasi akan menjadi $O(\log n)$.

Berdasarkan penjelasan di atas, ada dua hal yang menjadi dasar perhitungan kompleksitas algoritma yaitu total pekerjaan di setiap level ($O(n)$) dan jumlah level maksimal ($O(\log n)$). Kompleksitas waktu untuk algoritma ini untuk membangun Quadtree dengan pendekatan *Divide and Conquer* adalah:

$$O(n \log n)$$

BAB 6

Implementasi Bonus

Target Persentase Kompresi

Implementasi bonus pertama yang dikerjakan adalah pilihan untuk menggunakan target persentase kompresi. Fitur ini memungkinkan pengguna untuk menentukan target persentase kompresi gambar yang pengguna input, dengan nilai floating point (1.0 = 100%). Jika pengguna memberikan nilai 0, maka mode penyesuaian target persentase kompresi ini akan dinonaktifkan. Jika mengaktifkan mode ini, maka algoritma akan menyesuaikan threshold secara otomatis untuk mendapatkan target kompresi yang diinginkan. Penyesuaian threshold akan dilakukan secara dinamis, dengan tetap memberikan fleksibilitas dalam proses kompresi untuk memenuhi target efisiensi. Selain itu, ketika mode ini diaktifkan, minimum block size juga akan disesuaikan, sehingga masukan dari pengguna sebelumnya tidak digunakan.

Dalam program yang dibuat, perintah untuk memasukkan target persentase kompresi akan terlihat sebagai berikut:

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini):
```

Jika pengguna memberi masukan nilai floating point lebih dari 0.0, maka mode akan diaktifkan dan program akan memberitahu rentang target kompresi yang mungkin untuk dilakukan. Hal ini terjadi karena keterbatasan algoritma yang dibuat. Untuk mengerjakan bonus ini, minimum block size masukan pengguna sebelumnya tidak digunakan dan akan digantikan dengan 1. Kemudian, threshold akan dicari otomatis dengan mencoba threshold mana yang cocok untuk bisa mencapai target yang diinginkan pengguna. Akan tetapi, batasannya adalah, kasus ketika menggunakan minimum block size 1, dan threshold terkecil sekalipun (contoh 0 untuk Entropy) tetap tidak akan membuat persentase target kompresi di bawah batas tertentu. Begitu pula dengan threshold yang sangat besar sekalipun, tidak bisa membuat persentase target kompresi melebihi batas atas tertentu. Ambang atas dan bawah ini akan berbeda untuk setiap gambar yang dimasukan oleh pengguna.

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0,2  
Rentang target kompresi untuk metode ENTROPY adalah: 44,14% - 92,29%  
Input target kompresi tidak berada dalam rentang yang bisa dilakukan oleh program  
ini, sistem akan tetap mencoba.
```

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0,7  
Rentang target kompresi untuk metode VARIANCE adalah: 47,10% - 93,17%  
Target kompresi diaktifkan! Berusaha menargetkan kompresi ke: 70.0%
```

Jika masukan user berada pada rentang yang mungkin dilakukan program, maka hasil akhir gambar yang dikompresi akan sangat mendekati target di awal (lihat pengujian 7). Akan tetapi,

jika tidak memungkinkan, maka persentase kompresi akhir akan mengikuti batas bawah atau batas atas yang mungkin dilakukan oleh program.

Pencarian threshold otomatis ini dilakukan dengan menyimpan batas bawah dan batas atas threshold yang dimiliki suatu metode pencarian error. Kemudian, dalam 10 kali pengulangan, akan mencoba threshold tengah dari batas atas dan batas bawah, kemudian dicari compressed sizenya berapa menggunakan threshold tersebut, dan dibandingkan dengan original size. Jika ternyata target persentasenya kurang dari target, maka batas bawah akan diubah menjadi variabel mid sementara. Jika target persentase lebih dari target, maka batas atas akan diubah menjadi variabel mid sementara. Hal ini dilakukan sebanyak 10 kali hingga menemukan threshold yang sesuai dengan target yang diinginkan user.

```
Masukkan target persentase kompresi (0.0-1.0 | 0 untuk menonaktifkan mode ini): 0,7
Rentang target kompresi untuk metode MAD adalah: 47,10% - 93,17%
Target kompresi diaktifkan! Berusaha menargetkan kompresi ke: 70.0%

Masukkan path absolut untuk file hasil kompresi (contoh c:\folder\hasil.jpg): c:\Users\kefas\Documents\VSCode\ITB1
35\SMT4\STIMA\Tucil2\Tucil2_13523018_13523113\test\input\timurajetes.jpg
Gambar berhasil disimpan pada path yang dituju.
```

```
Statistik Kompresi Gambar
Waktu eksekusi: 0.407309 detik
Ukuran gambar sebelum: 390461 bytes
Ukuran gambar setelah: 117346 bytes
Persentase kompresi: 69.94680646722719%
Kedalaman pohon: 12
Banyak simpul pada pohon: 64189
```

Metode Pengukuran Error Structural Similarity Index (SSIM)

Implementasi bonus kedua adalah metode pengukuran error menggunakan SSIM. Pada dasarnya, metode SSIM ini membandingkan dua buah gambar dan dicari seberapa mirip kedua gambar tersebut berdasarkan tingkat kecerahan (luminous) dan kontras (RGB). Pada bonus ini, SSIM yang dilakukan adalah membandingkan gambar sebelum dikompresi, dengan gambar sesudah kompresi. Akan tetapi, karena program belum memiliki gambar setelah dikompresi untuk dibandingkan, maka program hanya membandingkan gambar original dan representasi area yang dianggap homogen untuk setiap kanal R, G dan B dan digabungkan hasilnya menjadi SSIM total. Oleh karena itu, SSIM yang digunakan adalah SSIM yang sudah disederhanakan.

SSIM berada pada rentang nilai 0-1, dengan 0 berarti tidak ada kemiripan struktural, dan 1 berarti kedua gambar identik secara struktural. Lihat Pengujian 6,7,8, dan 9 untuk hasil penggunaan metode SSIM.

Pembuatan GIF Proses Pembentukan Quadtree dalam Proses Kompresi

Implementasi bonus ketiga adalah pembuatan GIF proses pembentukan quadtree dalam proses kompresi suatu gambar. Cara kerja dari pembuatan GIF ini adalah melakukan rekonstruksi gambar untuk setiap level kedalaman quadtree yang terbentuk. Menggunakan for loop sesuai jumlah kedalaman quadtree, untuk setiap kedalaman akan dibuat satu frame dari gambar, dimana hanya node hingga level n yang dirender. Setiap frame adalah rekonstruksi dari gambar berdasarkan level kedalaman tertentu dari quadtree tersebut. Setelah itu, GIF dibuat menggunakan sebuah kelas GIFWriter yang dibuat dengan bantuan pustaka Java.

Akan tetapi, terdapat keterbatasan pada GIF yang dapat dibuat pada program ini. Ketika image original cukup besar, maka dapat terjadi error Java heap space, yaitu JVM kehabisan heap memory. Hal ini terjadi karena setiap frame sizenya cukup besar. Anggap setiap frame 1920x1080 pixel x 3 byte/pixel, sudah 6MB untuk satu frame. Jika ada 20 frame, sudah 120MB lebih, sehingga memori penuh.

Keterbatasan ini sebetulnya dapat diatasi dengan memperbesar heap Java menggunakan:

```
java -Xmx1G ProgramSaya
```

Akan tetapi, mengingat program ini akan digunakan oleh pengguna lain, dan mungkin saja pengguna lain tidak mengerti cara memperbesar heap Java. Maka pada program yang dibuat, pembuatan GIF dibatasi untuk gambar original dengan size yang tidak terlalu besar saja seperti:

```
boolean skipGIF = false;
if(image.getWidth() * image.getHeight() > 1920*1080){
    System.out.println("\nMaaf! Ukuran gambar terlalu besar untuk membuat GIF
proses kompresi karena memori Java tidak cukup. Kompresi akan tetap dijalankan tanpa
pembuatan GIF.");
    skipGIF = true;
}
```

hasil pembuatan GIF dapat dilihat pada bagian lampiran.

BAB 7

Kesimpulan

Melalui tugas kecil ini, telah berhasil diimplementasikan sebuah program kompresi gambar menggunakan strategi algoritma Divide and Conquer yang telah dipelajari pada mata kuliah IF2211 - Strategi Algoritma. Program yang dibuat mampu membagi gambar menjadi blok-blok yang lebih kecil berdasarkan keseragaman warna, menggunakan beberapa metode pengukuran error yaitu: Variance, Mean Absolute Deviation (MAD), Max Pixel Difference (MPD), Entropy, dan Structural Similarity Index (SSIM). Setiap metode pengukuran error memiliki karakteristik sendiri terhadap variasi warna dalam gambar. Selain itu, beberapa implementasi fitur tambahan juga telah dibuat dengan baik, meliputi target persentase kompresi, penggunaan SSIM, dan pembuatan GIF visualisasi proses pembentukan quadtree. Dengan demikian, strategi algoritma Divide and Conquer terbukti mampu untuk menyelesaikan permasalahan ini dengan kompleksitas waktu $O(n \log n)$ yang menunjukkan efisiensi yang cukup baik.

Lampiran

Source code program:

https://github.com/rakdaf08/Tucil2_13523018_13523113

Drive file .gif:

https://drive.google.com/drive/folders/10itbSpU_zeWKwR-OrwdFugvdDy8PpGvt?usp=sharing

atau lihat pada file DOCS berikut:

https://docs.google.com/document/d/1XrnrK8V_jTdNde9-c7r0MV3FRRDF1kySDDvczqtAwj4/edit?usp=sharing

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	✓	

Referensi

Spesifikasi Tugas Kecil 2: Kompresi Gambar Dengan Metode Quadtree.

Munir, Rinaldi. “Algoritma Divide and Conquer (Bagian 1).” *Homepage Rinaldi Munir*, 2025, [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf). Accessed 07 04 2025.