Tugas Kecil 2 IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree

Disusun Oleh:

Ahmad Wicaksono (13523121)

SEKOLAH TEKNIK ELEKTRO DAN INFORMARTIKA

INSTITUT TEKNOLOGI BANDUNG

2024

1. LATAR BELAKANG

1.1. Kompresi Gambar dengan Quadtree

Kompresi gambar merupakan salah satu teknik penting dalam pengolahan citra digital yang bertujuan untuk mengurangi ukuran file tanpa mengorbankan kualitas visual secara signifikan. Dalam konteks ini, struktur data Quadtree menjadi salah satu pendekatan yang menarik karena kemampuannya dalam merepresentasikan gambar berdasarkan tingkat homogenitas warna dalam blok-blok tertentu. Quadtree bekerja dengan cara membagi gambar menjadi empat bagian kuadran secara rekursif, dan hanya menyimpan informasi ketika blok-blok tersebut dianggap cukup seragam berdasarkan kriteria tertentu. Dengan demikian, Quadtree dapat memadatkan informasi visual secara efisien dan menghasilkan representasi gambar yang lebih ringan, yang sangat berguna dalam penyimpanan dan transmisi data gambar.

1.2. Algoritma Divide and Qonquer

Divide and Conquer merupakan paradigma algoritma yang membagi permasalahan besar menjadi submasalah yang lebih kecil, menyelesaikan submasalah tersebut secara rekursif, dan kemudian menggabungkan hasilnya untuk mendapatkan solusi akhir. Pendekatan ini sangat efektif dalam berbagai konteks komputasi karena dapat mengurangi kompleksitas waktu dan meningkatkan efisiensi proses. Dalam pengolahan gambar, Divide and Conquer cocok digunakan ketika kita ingin mengolah bagian-bagian gambar secara lokal, lalu mengintegrasikannya kembali ke dalam bentuk global. Salah satu kekuatan utama dari algoritma ini adalah kemampuannya untuk menangani data yang memiliki pola atau struktur lokal yang berulang atau homogen.

1.3. Metode Quadtree Menggunakan Algoritma Divide and Qonquer

Metode kompresi Quadtree merupakan implementasi langsung dari prinsip Divide and Conquer pada bidang pemrosesan gambar. Dengan menggunakan pendekatan ini, algoritma Quadtree akan membagi suatu gambar menjadi empat kuadran. Jika suatu kuadran tidak memenuhi kriteria homogenitas—misalnya, berdasarkan ambang batas variansi atau perbedaan warna—maka kuadran tersebut akan dibagi lagi menjadi empat bagian yang lebih kecil, dan proses ini berlanjut secara rekursif. Hanya blok-blok yang memenuhi ambang batas keseragaman yang akan disimpan sebagai simpul daun, sedangkan blok lain akan terus dibagi. Hal ini menciptakan struktur pohon yang mencerminkan distribusi informasi dalam gambar, serta memungkinkan kompresi data yang efisien dan adaptif terhadap kompleksitas visual suatu gambar.

2. IMPLEMENTASI ALGORITMA DALAM BAHASA JAVA

2.1. File CompressionPercentage.java

File CompressionPercentage.java berfungsi untuk menghitung persentase kompresi dari gambar sebelum dan sesudah proses kompresi berdasarkan ukuran file-nya. Nilai persentase yang dihasilkan menunjukkan seberapa besar data berhasil dikompresi dari bentuk aslinya.

Nama Fungsi	**Deskripsi**
calculateCompression	Menerima dua parameter berupa ukuran file asli (ori) dan ukuran file hasil kompresi (compress). Fungsi ini mengembalikan persentase kompresi dalam bentuk bulat (%), dengan membandingkan selisih ukuran terhadap ukuran asli. Jika ukuran asli adalah 0, maka fungsi akan mengembalikan 0 untuk menghindari pembagian dengan nol.

2.2. File ErrorCalculating.java

File ErrorCalculating.java berfungsi untuk menghitung nilai error antara gambar asli dan gambar hasil kompresi menggunakan beberapa metode yang dapat dipilih oleh pengguna. File ini berperan penting dalam evaluasi kualitas hasil kompresi.

Nama Fungsi	**Deskripsi**
calculateError	Fungsi utama yang memanggil metode perhitungan error berdasarkan parameter method. Mendukung 5 metode: Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy, dan Structural Similarity Index (SSIM).
variance	Menghitung rata-rata dari kuadrat selisih nilai piksel antara gambar asli dan hasil kompresi. Cocok untuk mengukur deviasi keseluruhan.
meanAbsoluteDeviation	Menghitung rata-rata dari nilai absolut selisih piksel, memberikan ukuran penyimpangan rata-rata dari gambar asli.
maxPixelDifference	Mengembalikan nilai maksimum dari selisih absolut piksel antara gambar asli dan hasil kompresi.

entropy	Menghitung tingkat ketidakteraturan (randomness) dari piksel dalam gambar asli, berguna sebagai baseline kompleksitas gambar.
structuralSimilarityIndex	Menghitung indeks kesamaan struktural antara gambar asli dan hasil kompresi untuk menilai kualitas visual secara subjektif.

2.3. File GifSequenceWriter.java

File GifSequenceWriter.java digunakan untuk membuat dan menulis animasi GIF dari serangkaian gambar BufferedImage. Kelas ini berguna untuk menyusun GIF yang menampilkan proses bertahap, seperti visualisasi kompresi Quadtree.

Nama Fungsi	**Deskripsi**
GifSequenceWriter	Konstruktor yang menginisialisasi writer GIF, metadata, dan konfigurasi seperti delay antar frame dan opsi loop (pengulangan).
getWriter	Mengambil ImageWriter yang mendukung format GIF dari ImageIO. Digunakan untuk menyiapkan penulisan berformat GIF.
getNode	Mencari atau membuat node metadata tertentu dalam pohon metadata gambar.
writeToSequence	Menambahkan satu gambar RenderedImage ke dalam urutan frame GIF.
close	Menyelesaikan proses penulisan GIF dan menutup stream output.

2.4. File ImageConverter.java

File ImageConverter.java digunakan untuk mengubah gambar berformat BufferedImage menjadi array dua dimensi yang merepresentasikan nilai grayscale dari tiap piksel. Konversi ini penting sebagai tahap awal sebelum pemrosesan kompresi menggunakan Quadtree.

Nama Fungsi	**Deskripsi**
toRGBArr	Mengubah gambar BufferedImage menjadi matriks yang berisi nilai RGB dari setiap piksel.

2.5. File InputImage.java

File InputImage.java digunakan untuk membaca gambar dari path yang diberikan dan mengubahnya menjadi objek BufferedImage untuk diproses lebih lanjut dalam sistem kompresi berbasis Quadtree.

Nama Fungsi	**Deskripsi**
readImage	Membaca file gambar dari path input. Fungsi ini memvalidasi keberadaan file, memastikan format gambar yang didukung, dan mengembalikan objek BufferedImage jika berhasil. Jika gagal, akan mengembalikan null dan menampilkan pesan error.

2.6. File QuadtreeCompression.java

File QuadtreeCompression.java menangani proses kompresi gambar berbasis Quadtree serta menyediakan visualisasi per level blok untuk animasi GIF.

Nama Fungsi	**Deskripsi**
QuadtreeCompression	Konstruktor untuk inisialisasi parameter awal kompresi.
compress	Mengambil ImageWriter yang mendukung format GIF dari ImageIO. Digunakan untuk menyiapkan penulisan berformat GIF.
divideAndConquer	Mencari atau membuat node metadata tertentu dalam pohon metadata gambar.
fillBlock	Menambahkan satu gambar RenderedImage ke dalam urutan frame GIF.
getAverageColor	Menyelesaikan proses penulisan GIF dan menutup stream output.
calculateError	Menghitung variance (keseragaman warna) dari blok gambar sebagai acuan homogenitas.
getCurrentCompression	Mengembalikan rasio kompresi saat ini berdasarkan jumlah node quadtree.
getExecutionTime	Waktu eksekusi kompresi dalam milidetik.
getTreeDepth	Kedalaman maksimum pohon Quadtree selama proses kompresi.
getNodeCount	Jumlah node yang terbentuk selama proses Quadtree.
compressAndCaptureFrames	Melakukan proses kompresi per level blok (1x1, 2x2, 4x4, dst) untuk animasi dan menyimpan frame di list.

isHomogeneousRGB	Mengecek apakah blok cukup homogen berdasarkan deviasi warna RGB.
getAverageColorRGB	Menghitung warna rata-rata dari blok (versi RGB, untuk visualisasi).
fillRegion	Mengisi region/frame dengan warna rata-rata (untuk animasi).
deepCopy	Membuat salinan independen dari gambar agar tidak mengubah gambar asli.

2.7. File SaveImage.java

File InputImage.java digunakan untuk membaca gambar dari path yang diberikan dan mengubahnya menjadi objek BufferedImage untuk diproses lebih lanjut dalam sistem kompresi berbasis Quadtree.

Nama Fungsi	**Deskripsi**
readImage	Membaca file gambar dari path input. Fungsi ini memvalidasi keberadaan file, memastikan format gambar yang didukung, dan mengembalikan objek BufferedImage jika berhasil. Jika gagal, akan mengembalikan null dan menampilkan pesan error.

2.8. File Main.java

File Main.java melakukan kompresi gambar menggunakan metode Quadtree dengan beberapa parameter yang dapat diatur pengguna, seperti metode perhitungan error (Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy, atau SSIM), nilai threshold, ukuran blok minimum, dan target persentase kompresi. Gambar input dibaca dan dikonversi menjadi array 2D grayscale untuk dianalisis, lalu dikompresi berdasarkan struktur pohon quadtree hingga memenuhi target kompresi atau batas error tertentu. Hasil kompresi disimpan dalam bentuk gambar serta, jika diinginkan, divisualisasikan dalam bentuk animasi GIF yang menunjukkan proses kompresi per level blok. Program juga menghitung nilai error hasil kompresi, ukuran file sebelum dan sesudah, serta informasi tambahan seperti waktu eksekusi, kedalaman pohon, dan jumlah simpul, yang semuanya ditampilkan secara naratif di konsol.

3. SOURCE CODE DALAM BAHASA JAVA

3.1. Repositori Github

Repositori program dapat diakses melalui tautan Github ini
https://github.com/sonix03/Tucil2_13523121

3.2. Source Code Program

3.2.1. CompressionPercentage.java

1. calculateCompression

```java
package src;

public class CompressionPercentage {
    public static long calculateCompression(long ori, long compress) {
        if (ori == 0L) {
            return 0L;
        } else {
            double percent = 1.0 - (double) compress / (double) ori;
            return Math.round(percent * 100.0); // dibulatkan ke persen
        }
    }
}
```

3.2.2. ErrorCalculating.java

1. calculateError

```java
public static double calculateError(int method, int[][] ori, int[][] compressed) {
    switch (method) {
        case 1: return variance(ori, compressed);
        case 2: return meanAbsoluteDeviation(ori, compressed);
        case 3: return maxPixelDifference(ori, compressed);
        case 4: return entropy(ori);
        case 5: return structuralSimilarityIndex(ori, compressed);
        default:
            throw new IllegalArgumentException(s:"Metode error tidak valid.");
    }
}
```

2. variance

```java
private static double variance(int[][][] ori, int[][][] compressed) {
    double sumR = 0, sumG = 0, sumB = 0;
    int height = ori.length;
    int width = ori[0].length;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            sumR += Math.pow(ori[i][j][0] - compressed[i][j][0], b:2);
            sumG += Math.pow(ori[i][j][1] - compressed[i][j][1], b:2);
            sumB += Math.pow(ori[i][j][2] - compressed[i][j][2], b:2);
        }
    }

    int count = height * width;
    return (sumR + sumG + sumB) / (3 * count);
}
```

3. meanAbsoluteDeviation

```java
private static double meanAbsoluteDeviation(int[][][] ori, int[][][] compressed) {
    double sumR = 0, sumG = 0, sumB = 0;
    int height = ori.length;
    int width = ori[0].length;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            sumR += Math.abs(ori[i][j][0] - compressed[i][j][0]);
            sumG += Math.abs(ori[i][j][1] - compressed[i][j][1]);
            sumB += Math.abs(ori[i][j][2] - compressed[i][j][2]);
        }
    }

    int count = height * width;
    return (sumR + sumG + sumB) / (3 * count);
}
```

4. maxPixelDifference

```java
private static double maxPixelDifference(int[][][] ori, int[][][] compressed) {
    int maxR = 0, maxG = 0, maxB = 0;
    int height = ori.length;
    int width = ori[0].length;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            maxR = Math.max(maxR, Math.abs(ori[i][j][0] - compressed[i][j][0]));
            maxG = Math.max(maxG, Math.abs(ori[i][j][1] - compressed[i][j][1]));
            maxB = Math.max(maxB, Math.abs(ori[i][j][2] - compressed[i][j][2]));
        }
    }

    return (maxR + maxG + maxB) / 3.0;
}
```

5. entropy

```java
private static double entropy(int[][][] ori) {
    double entropyR = 0, entropyG = 0, entropyB = 0;
    int[][] hist = new int[3][256];
    int height = ori.length;
    int width = ori[0].length;
    int pixelCount = height * width;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            hist[0][ori[i][j][0]]++;
            hist[1][ori[i][j][1]]++;
            hist[2][ori[i][j][2]]++;
        }
    }

    for (int i = 0; i < 256; i++) {
        if (hist[0][i] > 0) {
            double prob = (double) hist[0][i] / pixelCount;
            entropyR -= prob * (Math.log(prob) / Math.log(a:2));
        }
        if (hist[1][i] > 0) {
            double prob = (double) hist[1][i] / pixelCount;
            entropyG -= prob * (Math.log(prob) / Math.log(a:2));
        }
        if (hist[2][i] > 0) {
            double prob = (double) hist[2][i] / pixelCount;
            entropyB -= prob * (Math.log(prob) / Math.log(a:2));
        }
    }
    return (entropyR + entropyG + entropyB) / 3.0;
}
```

6. structuralSimilarityIndex

```java
private static double structuralSimilarityIndex(int[][][] ori, int[][][] compressed) {
    double sumR = 0, sumG = 0, sumB = 0;
    int height = ori.length;
    int width = ori[0].length;

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int r1 = ori[i][j][0], r2 = compressed[i][j][0];
            int g1 = ori[i][j][1], g2 = compressed[i][j][1];
            int b1 = ori[i][j][2], b2 = compressed[i][j][2];

            sumR += (2.0 * r1 * r2 + 1) / (r1 * r1 + r2 * r2 + 1);
            sumG += (2.0 * g1 * g2 + 1) / (g1 * g1 + g2 * g2 + 1);
            sumB += (2.0 * b1 * b2 + 1) / (b1 * b1 + b2 * b2 + 1);
        }
    }

    int count = height * width;
    return (sumR + sumG + sumB) / (3.0 * count);
}
```

3.2.3. GifSequenceWriter.java

1. GifSequenceWriter

```
public GifSequenceWriter(ImageOutputStream stream, int imageType, int delay, boolean loopF) throws IOException {
    this.stream = stream;
    this.writer = getWriter();
    this.param = writer.getDefaultWriteParam();

    ImageTypeSpecifier specific = ImageTypeSpecifier.createFromBufferedImageType(imageType);
    this.metadata = writer.getDefaultImageMetadata(specific, param);

    String formatName = metadata.getNativeMetadataFormatName();
    IIOMetadataNode root = (IIOMetadataNode) metadata.getAsTree(formatName);

    IIOMetadataNode graphNode = getNode(root, nodeName:"GraphicControlExtension");
    graphNode.setAttribute(name:"disposalMethod", value:"none");
    graphNode.setAttribute(name:"userInputFlag", value:"FALSE");
    graphNode.setAttribute(name:"transparentColorFlag", value:"FALSE");
    graphNode.setAttribute(name:"delayTime", Integer.toString(delay / 10));
    graphNode.setAttribute(name:"transparentColorIndex", value:"0");

    IIOMetadataNode ensNode = getNode(root, nodeName:"ApplicationExtensions");
    IIOMetadataNode appNode = new IIOMetadataNode(nodeName:"ApplicationExtension");

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

    int loop = loopF ? 0 : 1;
    byte[] loopBytes = new byte[]{0x1, (byte) (loop & 0xFF), (byte) ((loop >> 8) & 0xFF)};
    appNode.setUserObject(loopBytes);
    ensNode.appendChild(appNode);
    root.appendChild(ensNode);
```

2. getWriter

```
private static ImageWriter getWriter() {
    Iterator<ImageWriter> writers = ImageIO.getImageWritersBySuffix(fileSuffix:"gif");
    if (!writers.hasNext()) throw new IllegalStateException(s:"Gaada GIF");
    return writers.next();
}
```

3. getNode

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

4. writeToSequence

```
public void writeToSequence(RenderedImage image) throws IOException {
    writer.writeToSequence(new IIOImage(image, thumbnails:null, metadata), param);
}
```

5. close

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

3.2.4. ImageConverter.java

1. toRGBArr

```java
public static int[][][] toRGBArr(BufferedImage image) {
    int width = image.getWidth();
    int height = image.getHeight();
    int[][][] rgbArr = new int[height][width][3];

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int rgb = image.getRGB(x, y);
            int r = (rgb >> 16) & 0xFF;
            int g = (rgb >> 8) & 0xFF;
            int b = rgb & 0xFF;

            rgbArr[y][x][0] = r;
            rgbArr[y][x][1] = g;
            rgbArr[y][x][2] = b;
        }
    }

    return rgbArr;
}
```

3.2.5. InputImage.java

1. readImage

```java
public static BufferedImage readImage(String inputPath) {
    try {
        File file = new File(inputPath);
        if (!file.exists()) {
            System.out.println("File tidak ditemukan: " + inputPath);
            return null;
        }

        BufferedImage image = ImageIO.read(file);
        if (image == null) {
            System.out.println(x:"Format gambar tidak didukung");
            return null;
        }

        System.out.println("bisa baca. Lebar: " + image.getWidth() + ", Tinggi: " + image.getHeight());
        return image;
    } catch (IOException e) {
        System.err.println("Error baca gambar: " + e.getMessage());
        return null;
    }
}
```

3.2.6. QuadtreeCompression.java

1. QuadtreeCompression

```
public QuadtreeCompression(BufferedImage image, double threshold, int minBlockSize,
double targetCompression) {
    this.image = image;
    this.threshold = threshold;
    this.minBlockSize = minBlockSize;
    this.targetCompression = targetCompression;
}
```

2. compress

```
public BufferedImage compress() {
    long startTime = System.currentTimeMillis();
    int width = image.getWidth();
    int height = image.getHeight();
    BufferedImage compressedImage = new BufferedImage(width, height, image.getType());
    Graphics2D g2d = compressedImage.createGraphics();

    // Mulai rekursi quadtree
    divideAndConquer(x:0, y:0, width, height, g2d, depth:0);
    g2d.dispose();

    executionTime = System.currentTimeMillis() - startTime;
    return compressedImage;
}
```

3. divideAndQonquer

```
private void divideAndConquer(int x, int y, int width, int height, Graphics2D g2d, int depth) {
    boolean reachMinBlock = (width <= minBlockSize || height <= minBlockSize);
    boolean reachTargetCompression = (targetCompression > 0 && getCurrentCompression() >= targetCompression);

    if (reachMinBlock || reachTargetCompression) {
        fillBlock(x, y, width, height, g2d);
        return;
    }

    // Langsung bagi 4 tanpa cek threshold
    int halfWidth = width / 2;
    int halfHeight = height / 2;

    divideAndConquer(x, y, halfWidth, halfHeight, g2d, depth + 1);                          // a
    divideAndConquer(x + halfWidth, y, width - halfWidth, halfHeight, g2d, depth + 1);      // b
    divideAndConquer(x, y + halfHeight, halfWidth, height - halfHeight, g2d, depth + 1);    // c
    divideAndConquer(x + halfWidth, y + halfHeight, width - halfWidth, height - halfHeight, g2d, depth + 1); // d

    if (captureFrames) {
        frames.add(deepCopy(workingImage));
    }

    nodeCount += 4;
    treeDepth = Math.max(treeDepth, depth);
}
```

4. fillBlock

```
private void fillBlock(int x, int y, int width, int height, Graphics2D g2d) {
    Color avgColor = getAverageColor(x, y, width, height);
    g2d.setColor(avgColor);
    g2d.fillRect(x, y, width, height);

    if (captureFrames) {
        Graphics2D gWork = workingImage.createGraphics();
        gWork.setColor(avgColor);
        gWork.fillRect(x, y, width, height);
        gWork.dispose();
        frames.add(deepCopy(workingImage));
    }
}
```

5.getAverageColor

```
private Color getAverageColor(int x, int y, int width, int height) {
    int r = 0, g = 0, b = 0, count = 0;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            r += c.getRed();
            g += c.getGreen();
            b += c.getBlue();
            count++;
        }
    }
    return new Color(r / count, g / count, b / count);
}
```

6. calculateError

```
private double calculateError(int x, int y, int width, int height) {
    double sum = 0, sumA = 0;
    int count = 0;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            int gray = new Color(image.getRGB(i, j)).getRed();
            sum += gray;
            sumA += gray * gray;
            count++;
        }
    }

    double mean = sum / count;
    return (sumA / count) - (mean * mean); // Variance sebagai ukuran error
}
```

7. getCurrentCompression

```
private double getCurrentCompression() {
    int countPix = image.getWidth() * image.getHeight();
    return (double) nodeCount / countPix;
}
```

8. getExecutionTime

```
}
    return executionTime;
public long getExecutionTime() {
```

9. getTreeDepth

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

10. getNodeCount

```
public int getNodeCount() {
    return nodeCount;
}
```

11. compressAndCaptureFrames

```java
public List<BufferedImage> compressAndCaptureFrames() {
    BufferedImage framesImage = deepCopy(image);
    int size = 2;
    while (size <= Math.min(framesImage.getWidth(), framesImage.getHeight())) {
        BufferedImage frame = deepCopy(framesImage);
        for (int y = 0; y < frame.getHeight(); y += size) {
            for (int x = 0; x < frame.getWidth(); x += size) {
                int blockW = Math.min(size, frame.getWidth() - x);
                int blockH = Math.min(size, frame.getHeight() - y);
                if (isHomogeneousRGB(frame, x, y, blockW, blockH)) {
                    Color avg = getAverageColorRGB(frame, x, y, blockW, blockH);
                    fillRegion(frame, x, y, blockW, blockH, avg);
                }
            }
        }
        frames.add(frame);
        size *= 2;
    }
    return frames;
}
```

12. isHomogeneousRGB

```java
private boolean isHomogeneousRGB(BufferedImage image, int x, int y, int width, int height) {
    Color avg = getAverageColorRGB(image, x, y, width, height);
    double sum = 0;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            double dif = Math.sqrt(Math.pow(c.getRed() - avg.getRed(), b:2)
                    + Math.pow(c.getGreen() - avg.getGreen(), b:2)
                    + Math.pow(c.getBlue() - avg.getBlue(), b:2));
            sum += dif;
        }
    }
    double res = sum / (width * height);
    return res < threshold;
}
```

13. getAverageColorRGB

```java
private Color getAverageColorRGB(BufferedImage image, int x, int y, int width, int height) {
    long r = 0, g = 0, b = 0;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            Color c = new Color(image.getRGB(i, j));
            r += c.getRed();
            g += c.getGreen();
            b += c.getBlue();
        }
    }
    int countPix = width * height;
    return new Color((int)(r / countPix), (int)(g / countPix), (int)(b / countPix));
}
```

14. fillRegion

```java
private void fillRegion(BufferedImage image, int x, int y, int width, int height, Color color) {
    Graphics2D g = image.createGraphics();
    g.setColor(color);
    g.fillRect(x, y, width, height);
    g.dispose();
}
```

15. deepCopy

```java
private BufferedImage deepCopy(BufferedImage bi) {
    BufferedImage cpy = new BufferedImage(bi.getWidth(), bi.getHeight(), BufferedImage.TYPE_INT_RGB);
    Graphics g = cpy.getGraphics();
    g.drawImage(bi, x:0, y:0, observer:null);
    g.dispose();
    return cpy;
}
```

3.2.7. SaveImage.java

1. writeImage

```java
public static void writeImage(BufferedImage image, String outputPath) {
    try {
        File output = new File(outputPath);
        ImageIO.write(image, formatName:"jpg", output);
        System.out.println("Gambar berhasil disimpan di: " + outputPath);
    } catch (IOException e) {
        System.err.println("Error menyimpan gambar: " + e.getMessage());
    }
}
```

3.2.8. Main.java

1. Main

5. INPUT DAN OUTPUT PROGRAM

Bonus: Untuk output yang dalam bentuk GIF bisa di cek di dalam folder output pada program.

5.1. Test Case 1 (Variance)

Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: 1.jpg
bisa baca. Lebar: 4000, Tinggi: 6000
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 1
Masukkan Threshold: 100
Masukkan ukuran blok minimum: 30
Masukkan target kompresi (0.0 - 1.0): 0.5
Masukkan alamat absolut gambar hasil kompresi: variance.jpg
kompresi gambar.....
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/variance.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 701 ms
Ukuran gambar sebelum: 2701350 bytes
Ukuran gambar setelah: 1118455 bytes
Persentase kompresi: 59.0%
Kedalaman pohon: 8
Banyak simpul pada pohon: 87380
Nilai error setelah kompresi: 385.4752541666667
Gambar hasil kompresi disimpan di: variance.jpg
```

variance.jpg:

5.2. Test Case 2 (MAD)

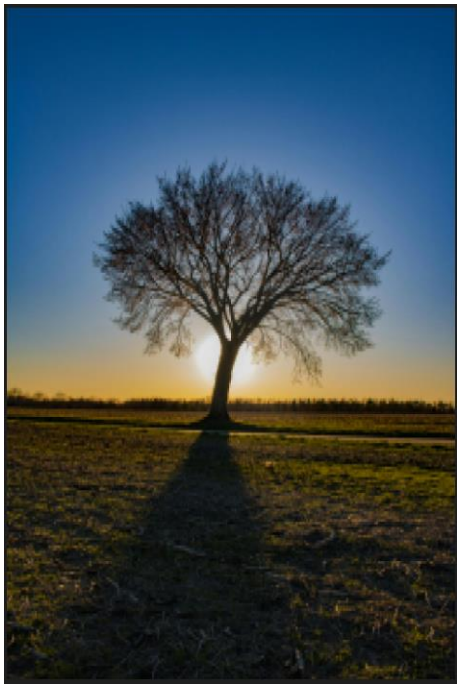
Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: 1.jpg
bisa baca. Lebar: 4000, Tinggi: 6000
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 2
Masukkan Threshold: 100
Masukkan ukuran blok minimum: 20
Masukkan target kompresi (0.0 - 1.0): 0.4
Masukkan alamat absolut gambar hasil kompresi: MAD.jpg
kompresi gambar.....
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/MAD.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 647 ms
Ukuran gambar sebelum: 2701350 bytes
Ukuran gambar setelah: 1118455 bytes
Persentase kompresi: 59.0%
Kedalaman pohon: 8
Banyak simpul pada pohon: 87380
Nilai error setelah kompresi: 10.359998694444444
Gambar hasil kompresi disimpan di: MAD.jpg
```

MAD.jpg:



5.3. Test Case 3 (MPD)

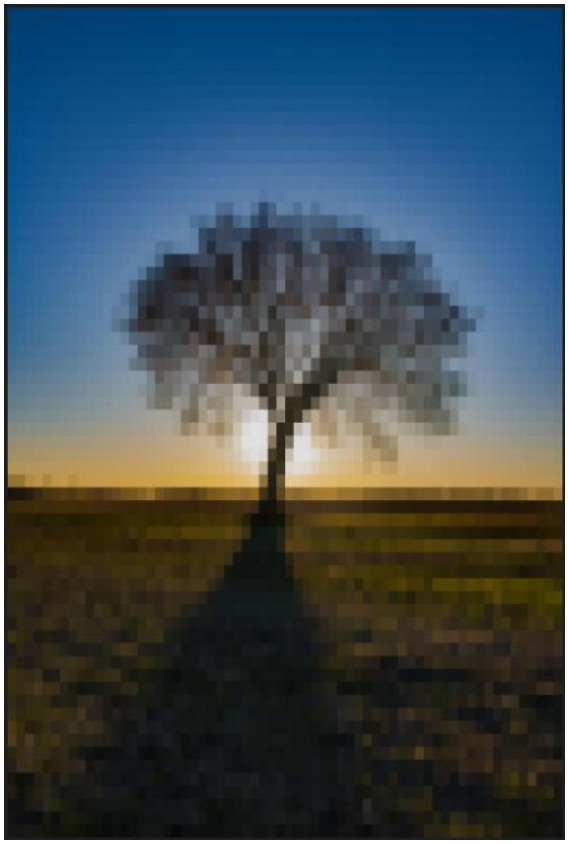
Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: 1.jpg
bisa baca. Lebar: 4000, Tinggi: 6000
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 3
Masukkan Threshold: 100
Masukkan ukuran blok minimum: 80
Masukkan target kompresi (0.0 - 1.0): 0.7
Masukkan alamat absolut gambar hasil kompresi: MPD.jpg
kompresi gambar......
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/MPD.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 643 ms
Ukuran gambar sebelum: 2701350 bytes
Ukuran gambar setelah: 587697 bytes
Persentase kompresi: 78.0%
Kedalaman pohon: 6
Banyak simpul pada pohon: 5460
Nilai error setelah kompresi: 224.0
Gambar hasil kompresi disimpan di: MPD.jpg
```

MPD.jpg:



5.4. Test Case 4 (Entropy)

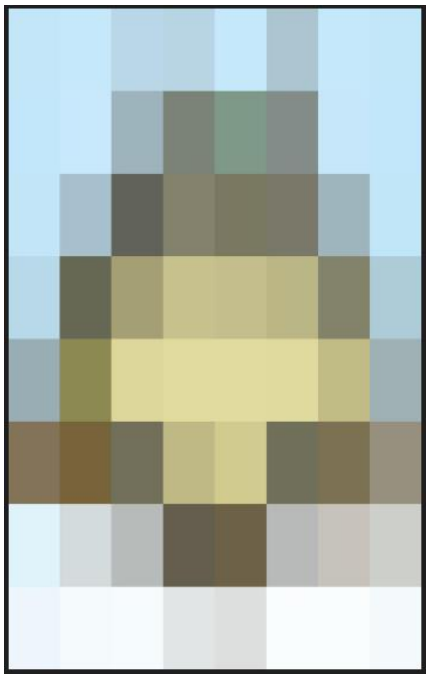
Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: totoro.jpg
bisa baca. Lebar: 599, Tinggi: 960
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 4
Masukkan Threshold: 150
Masukkan ukuran blok minimum: 100
Masukkan target kompresi (0.0 - 1.0): 0.5
Masukkan alamat absolut gambar hasil kompresi: entropy.jpg
kompresi gambar......
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/entropy.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 63 ms
Ukuran gambar sebelum: 53103 bytes
Ukuran gambar setelah: 16015 bytes
Persentase kompresi: 70.0%
Kedalaman pohon: 3
Banyak simpul pada pohon: 84
Nilai error setelah kompresi: 4.966072081178306
Gambar hasil kompresi disimpan di: entropy.jpg
```

entropy.jpg:



5.5. Test Case 5 (SSIM)

Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: totoro.jpg
bisa baca. Lebar: 599, Tinggi: 960
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 5
Masukkan Threshold: 10
Masukkan ukuran blok minimum: 5
Masukkan target kompresi (0.0 - 1.0): 0.5
Masukkan alamat absolut gambar hasil kompresi: SSIM.jpg
kompresi gambar.....
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/SSIM.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 58 ms
Ukuran gambar sebelum: 53103 bytes
Ukuran gambar setelah: 45328 bytes
Persentase kompresi: 15.0%
Kedalaman pohon: 7
Banyak simpul pada pohon: 21844
Nilai error setelah kompresi: 0.9654890114639108
Gambar hasil kompresi disimpan di: SSIM.jpg
```

SSIM.jpg:

5.6. Test Case 6 (Variance2)

Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: totoro.jpg
bisa baca. Lebar: 599, Tinggi: 960
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 1
Masukkan Threshold: 100
Masukkan ukuran blok minimum: 30
Masukkan target kompresi (0.0 - 1.0): 0.5
Masukkan alamat absolut gambar hasil kompresi: variance2.jpg
kompresi gambar.....
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/variance2.jpg
Masukkan nama file GIF (kosongin untuk skip):
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 58 ms
Ukuran gambar sebelum: 53103 bytes
Ukuran gambar setelah: 31612 bytes
Persentase kompresi: 40.0%
Kedalaman pohon: 5
Banyak simpul pada pohon: 1364
Nilai error setelah kompresi: 1279.9386523836024
Gambar hasil kompresi disimpan di: variance2.jpg
```

Variance2.jpg:

5.7. Test Case 7 (GIF)

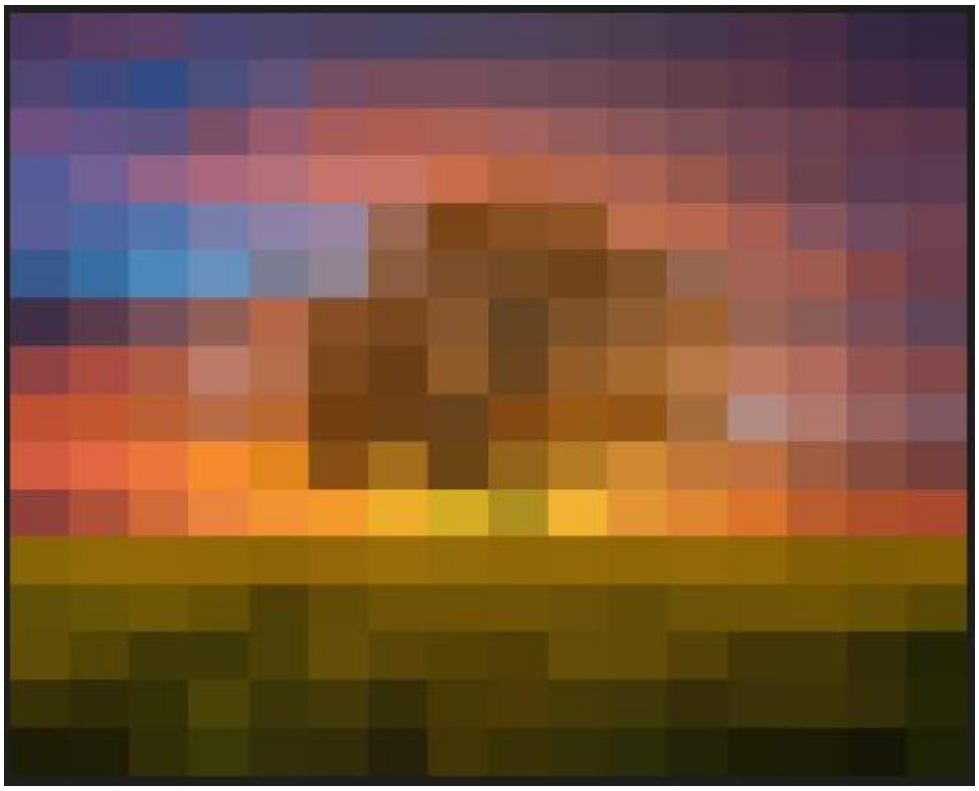
Input:

```
Masukkan alamat absolut gambar yang akan dikompresi: test.jpg
bisa baca. Lebar: 612, Tinggi: 490
Metode:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Pilih metode perhitungan error: 1
Masukkan Threshold: 150
Masukkan ukuran blok minimum: 50
Masukkan target kompresi (0.0 - 1.0): 0.5
Masukkan alamat absolut gambar hasil kompresi: image.jpg
kompresi gambar.....
Gambar berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/image.jpg
Masukkan nama file GIF (kosongin untuk skip): GIF.gif
GIF animasi berhasil disimpan di: C:/Documents/a_semester_4/Stima/Lanjut2/Tucil-2-Gabung/test/output/GIF.gif
berhasil!!!
```

Output:

```
--- Hasil Kompresi ---
Waktu eksekusi: 45 ms
Ukuran gambar sebelum: 81292 bytes
Ukuran gambar setelah: 16344 bytes
Persentase kompresi: 80.0%
Kedalaman pohon: 4
Banyak simpul pada pohon: 340
Nilai error setelah kompresi: 686.9446000622471
Gambar hasil kompresi disimpan di: image.jpg
```

image.jpg:



GIF.gif:

6. ANALISIS ALGORITMA

Algoritma QuadtreeCompression menggunakan pendekatan divide-and-conquer untuk membagi gambar menjadi blok-blok kecil secara rekursif, hingga mencapai ukuran minimum atau target kompresi tertentu. Kompleksitas waktunya berada di kisaran $O(n^2 \log n)$ karena setiap pembagian membagi blok menjadi empat bagian, dan operasi pewarnaan membutuhkan traversing seluruh piksel. Keunggulan utamanya terletak pada modularitas dan dukungan visualisasi melalui animasi frame (GIF) yang menunjukkan proses kompresi per level blok. Selain itu, pengguna dapat mengontrol tingkat kompresi melalui parameter minBlockSize, threshold, dan targetCompression, yang membuatnya fleksibel untuk berbagai kebutuhan.

Namun, algoritma ini masih memiliki beberapa kekurangan, terutama pada metode compress() yang membagi blok tanpa mempertimbangkan homogenitas warna, sehingga dapat menghasilkan struktur pohon yang tidak efisien dan memperbesar ukuran kompresi. Di sisi lain, metode compressAndCaptureFrames() yang justru mempertimbangkan threshold homogenitas tidak menggunakan pendekatan rekursif, menciptakan inkonsistensi logika. Penggunaan workingImage juga berpotensi menyebabkan bug karena tidak selalu diinisialisasi. Untuk hasil kompresi yang optimal dan lebih realistis, akan lebih baik jika logika threshold diterapkan secara konsisten dalam metode rekursif compress().

7. LAMPIRAN

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	v	
2. Program berhasil dijalankan	v	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentuka	v	
4. Mengimplementasi seluruh metode perhitungan error wajib	v	
5. **[Bonus]** Implementasi persentase kompresi sebagai parameter tambahan	v	
6. **[Bonus]** Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	v	

7. **[Bonus]** Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	v	
8. Program dan laporan dibuat (kelompok) sendiri	v	

8. REFERENSI

- https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tucil2-Stima-2025.pdf
- https://en.wikipedia.org/wiki/Quadtree
- https://romanglushach.medium.com/what-is-a-quadtree-and-how-it-works-6286791fb46a
- https://en.wikipedia.org/wiki/Divide-and-conquer_algorithm