

IF2211 Strategi Algoritma
Kompresi Gambar dengan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma
Semester II Tahun Akademik 2024/2025



Disusun oleh:

Amira Izani - 13523143

Natalia Desiany Nursimin - 13523157

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG

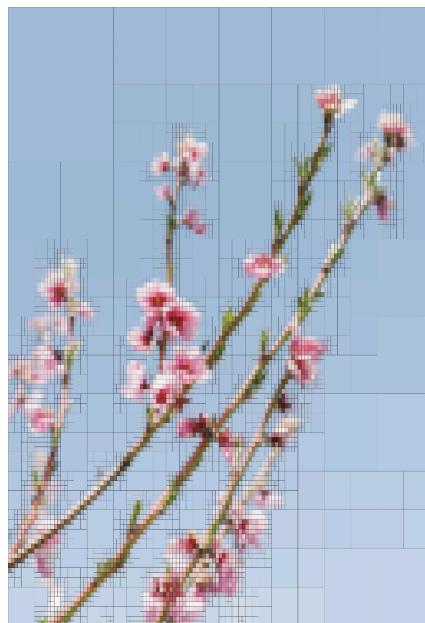
2025

DAFTAR ISI

BAB I DESKRIPSI MASALAH.....	3
BAB II TEORI SINGKAT.....	7
BAB III ALGORITMA DIVIDE AND CONQUER.....	8
BAB IV SOURCE CODE.....	10
4.1. Library yang Digunakan.....	10
4.2. Struktur Direktori Program.....	11
4.3. Struktur Data Program.....	13
4.3.1. QuadTree.java.....	13
A. Kode Program QuadTree.java.....	13
B. Atribut QuadTree.....	16
C. Metode QuadTree.....	17
4.3.2. Node.java.....	18
A. Kode Program Node.java.....	18
B. Atribut Node.....	19
C. Metode QuadTree.....	19
4.3.3. ImageCompressor.java.....	19
A. Kode Program ImageCompressor.java.....	20
B. Atribut ImageCompressor.....	23
4.3.4. ErrorCalculator.java.....	23
B. Atribut ErrorCalculator.....	28
C. Metode ErrorCalculator.....	28
4.3.5. GIFExporter.java.....	29
A. Kode Program GIFExporter.java.....	29
B. Atribut GIFExporter.....	30
C. Metode GIFExporter.....	30
4.3.6. Main.java.....	31
A. Kode Main.java.....	31
B. Atribut Main.....	33
C. Metode Main.....	33
BAB V HASIL DAN ANALISIS PENGUJIAN.....	34
5.1. Test Case 1 (Variance).....	34
5.2. Test Case 2 (MAD).....	34
5.3. Test Case 3 (Max pixel difference).....	35
5.4. Test Case 4 (Entropy).....	36
5.5. Test Case 5 (SSIM).....	36
5.6. Test Case 6 (Variance with Target Compression).....	37
5.7. Test Case 7 (SSIM with Target Compression).....	38

5.8. Test Case 8 (Variance with Target compression and different image).....	39
5.9. Test Case 9 (Entropy with high quality image).....	40
5.10. Test Case 10 (Entropy with low quality image).....	41
5.11. Analisis Test Case.....	42
BAB VI IMPLEMENTASI BONUS.....	43
 6.1. SSIM.....	43
6.1.1. Kode Pengimplementasian SSIM.....	43
6.1.2. Penjelasan Kode SSIM.....	44
 6.2. Target Kompresi.....	46
6.2.1. Kode Pengimplementasian Target Kompresi.....	46
6.2.2. Penjelasan Kode Target Kompresi.....	48
 6.3 GIF Visualisasi.....	49
6.3.1. Kode Pengimplementasian GIF Visualisasi.....	49
6.3.2. Penjelasan Kode GIF Visualisasi.....	49
BAB VII KESIMPULAN DAN SARAN.....	51
LAMPIRAN.....	52
Tautan Repository GitHub:.....	52
Tautan Drive Berisi Gambar Test Case:.....	52
Tabel Checklist Spesifikasi Program:.....	52

BAB I DESKRIPSI MASALAH



Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

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

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan

pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

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

Tabel 1. Metode Pengukuran Error

Metode	Formula
<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok

Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok $P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c
Entropy	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok $P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)
[Bonus] <u>Structural Similarity Index (SSIM)</u> <u>(Referensi tambahan)</u>	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

2. Threshold (Ambang Batas)

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

3. Minimum Block Size (Ukuran Blok Minimum)

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

4. Compression Percentage (Percentase Kompresi) [BONUS]

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

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

Berdasarkan permasalahan di atas, kami membuat sebuah program CLI dalam bahasa Java yang mengimplementasikan kompresi gambar berbasis Quadtree dengan pendekatan *divide and conquer*. Program ini menggunakan berbagai parameter seperti yang telah ditunjukkan sebelumnya, sebagai masukan dan menghasilkan keluaran berupa gambar hasil kompresi beserta data statistik yang relevan. Berikut adalah alur input-output pada program yang telah dibuat:

1. [INPUT] **alamat absolut** gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai *input*).
3. [INPUT] ambang batas (pastikan *range* nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi (*floating number*, $1.0 = 100\%$), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
6. [INPUT] **alamat absolut** gambar hasil kompresi.
7. [INPUT] **alamat absolut** gif (bonus).
8. [OUTPUT] waktu eksekusi.
9. [OUTPUT] ukuran gambar sebelum.
10. [OUTPUT] ukuran gambar setelah.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon.
13. [OUTPUT] banyak simpul pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

BAB II TEORI SINGKAT

Pada pengembangan aplikasi yang berkaitan dengan pengolahan gambar, efisiensi dan kecepatan pemrosesan merupakan dua aspek yang sangat penting. Seiring dengan semakin tingginya resolusi dan kompleksitas sebuah gambar, diperlukan juga teknik kompresi yang dapat mengurangi ukuran berkas dengan tetap mempertahankan kualitas visual yang baik. Salah satu pendekatan yang dapat dimanfaatkan untuk mengatasi permasalahan ini adalah dengan menggunakan struktur data Quadtree dan strategi pemecahan masalah *divide and conquer*, untuk membangun sebuah program yang efektif.

Quadtree merupakan sebuah struktur data berbentuk pohon yang membagi ruang dua dimensi secara rekursif ke dalam empat bagian kuadran. Dalam konteks kompresi gambar, Quadtree dapat digunakan untuk memecah gambar menjadi blok-blok yang lebih kecil dan mengelompokkannya berdasarkan kemiripan warna pikselnya. Proses ini dapat dilakukan secara rekursif dan akan terus berlanjut hingga dua kondisi tercapai, yaitu blok dianggap cukup seragam atau telah mencapai ukuran minimum yang dipilih. Blok-blok yang sudah memenuhi kriteria akan kemudian direpresentasikan sebagai simpul-simpul daun dalam struktur pohon Quadtree. Pendekatan metode ini memungkinkan penghilangan pengulangan yang tidak diperlukan pada bagian-bagian gambar yang seragam, sehingga menghasilkan kompresi yang efisien.

Penilaian terhadap keseragaman blok ini dilakukan dengan menghitung nilai error melalui berbagai metode pengukuran seperti *Variance*, *Mean Absolute Deviation (MAD)*, *Max Pixel Difference* dan *Entropy*. Selain itu, terdapat juga beberapa fitur tambahan yang dapat diterapkan dengan menggunakan *Structural Similarity Index (SSIM)*, yaitu sebuah metode penilaian berbasis persepsi manusia terhadap kualitas visual. Setiap metode ini dapat digunakan untuk memberikan pendekatan berbeda dalam menilai seberapa homogen suatu blok gambar.

Proses-proses ini dijalankan dengan menggunakan algoritma *divide and conquer*. *Divide and conquer* adalah sebuah strategi pemecahan masalah dengan cara membagi masalah besar menjadi bagian-bagian kecil (*divide*), kemudian menyelesaikan bagian-bagian tersebut secara independen (*conquer*), dan terakhir menggabungkan hasilnya untuk membentuk solusi akhir (*combine*). Dalam penggunaanya pada Quadtree, gambar akan dibagi menjadi empat sub-blok, kemudian masing-masing sub-blok akan diproses secara rekursif hingga diperoleh representasi akhir dari gambar yang telah berhasil dikompresi.

BAB III ALGORITMA DIVIDE AND CONQUER

Algoritma *divide and conquer* diimplementasikan pada proses kompresi gambar menggunakan struktur data Quadtree. Algoritma pada program ini dibangun dengan menggunakan pendekatan rekursif, di mana gambar akan pertama secara bertahap dibagi menjadi blok-blok kecil berdasarkan tingkat keseragaman warna. Algoritma *divide and conquer* pada program ini akan bekerja dengan membagi gambar menjadi empat bagian, memproses setiap bagian secara terpisah, lalu menyatukannya kembali untuk membentuk hasil akhir. Berikut adalah langkah-langkah penerapan algoritma *divide and conquer* pada program:

1. Divide (Untuk pembagian blok)

Strategi algoritma *divide and conquer* pada program diawali dengan pertama menganggap seluruh gambar sebagai satu blok utuh yang akan dianalisis. Pada tahap ini, blok tersebut akan dievaluasi untuk menentukan apakah perlu dibagi menjadi bagian-bagian yang lebih kecil. Evaluasi dilakukan melalui fungsi `shouldSplit(x, y, width, height, minBlockSize)` pada program dengan mempertimbangkan dua kriteria utama, yaitu:

- Apakah ukuran blok cukup besar untuk dibagi lebih lanjut (tidak lebih kecil dari ukuran minimum yang ditentukan).
- Apakah nilai error yang dihitung berdasarkan metode tertentu (Variance, MAD, MaxDiff, Entropy, atau SSIM) melebihi threshold yang telah ditentukan.

Jika kedua syarat tersebut terpenuhi, maka blok dibagi menjadi empat sub-blok kuadrat, yaitu kiri atas, kanan atas, kiri bawah dan kanan bawah. Setelah itu, masing-masing sub-blok akan dijadikan simpul anak (*children*) pada struktur pohon.

Kemudian, untuk setiap sub-blok yang dihasilkan, algoritma akan memanggil fungsi `buildRecursive(...)` secara rekursif pada keempat sub-blok. Setiap sub-blok akan diuji dan dibagi kembali sesuai dengan kriteria yang sama, sehingga proses ini berjalan secara hierarkis dan terus menurun ke level yang lebih dalam. Proses rekursif ini berlangsung hingga tidak ada lagi blok yang memenuhi syarat untuk dibagi.

2. Conquer (Untuk menyelesaikan sub-masalah)

Jika suatu blok tidak memenuhi syarat untuk dibagi, maka blok tersebut dianggap sebagai blok seragam. Di tahap ini, blok-blok tersebut dikonversi menjadi simpul daun (leaf node) dalam struktur Quadtree. Setiap leaf node menyimpan informasi posisi dan ukuran blok, serta warna rata-rata blok tersebut yang dihitung berdasarkan nilai RGB dari semua piksel di dalamnya.

Simpul-simpul daun ini akan digunakan untuk mewakili bagian gambar yang akan direkonstruksi pada hasil akhir.

3. *Combine* (Untuk melakukan pembangunan kembali gambar)

Setelah seluruh blok pada gambar selesai diproses, informasi dari semua simpul-simpul daun akan digunakan untuk menyusun ulang gambar hasil kompresi. Proses ini dilakukan dalam fungsi `reconstructImage()`. Setelah itu, algoritma program akan menggabungkan seluruh solusi dari setiap sub-masalah untuk membentuk solusi akhir berupa gambar terkompresi.

BAB IV SOURCE CODE

Program pada tugas kecil kompresi gambar menggunakan metode Quadtree ini ditulis dalam bahasa pemrograman Java. Program ini mengimplementasikan algoritma *divide and conquer* melalui struktur data Quadtree untuk melakukan kompresi gambar secara adaptif dan efisien. Program ini memungkinkan pengguna untuk mengompres gambar menggunakan berbagai metode penghitungan error, seperti *Variance*, *MAD*, *Max Pixel Difference*, *Entropy*, dan *SSIM*. Program akan membaca gambar berformat RGB, membaginya secara rekursif ke dalam blok-blok lebih kecil berdasarkan tingkat keseragaman warna, lalu menyimpan hasilnya sebagai gambar terkompresi dan visualisasi GIF (opsional). Berikut merupakan penjelasan source code pada program:

4.1. Library yang Digunakan

Library	Fungsi
<code>java.awt.image.BufferedImage</code>	Berfungsi untuk membaca dan manipulasi data piksel gambar
<code>java.awt.Color</code>	Berfungsi untuk mengelola dan menghitung komponen warna RGB dari setiap piksel pada gambar.
<code>javax.imageio.ImageIO</code>	Berfungsi untuk memfasilitasi proses pembacaan (input) dan penulisan (output) file gambar dalam berbagai format, seperti PNG, JPG dan JPEG.
<code>java.io.File</code>	Berfungsi untuk mendefinisikan dan mengakses jalur file input dan output.
<code>java.io.IOException</code>	Berfungsi untuk menangani pengecualian (exception) saat terjadi kesalahan input/output.
<code>javax.imageio.stream.ImageOutputStream</code>	Berfungsi untuk mengatur aliran data saat menyimpan gambar dalam format GIF secara berurutan.

javax.imageio.metadata.IIOMetadata	Berfungsi untuk menyimpan metadata tambahan dari gambar, termasuk pengaturan animasi dan delay antar frame dalam file GIF.
javax.imageio.metadata.IIOMetadataNode	Berfungsi untuk membentuk struktur hierarki metadata untuk mendeskripsikan properti frame dalam animasi GIF.
javax.imageio.ImageTypeSpecifier	Berfungsi untuk menentukan tipe gambar yang akan digunakan dalam proses encoding GIF.
java.util.Scanner	Berfungsi untuk membaca input dari pengguna melalui antarmuka command line.
java.util.List	Berfungsi untuk menyimpan daftar <i>children nodes</i> dalam struktur data pohon secara dinamis.

4.2. Struktur Direktori Program

Program yang dibuat memiliki struktur direktori sebagai berikut:

```
Tucil2_13523143_13523157/ (Root Directory)
|
└── src/
    ├── ErrorCalculator.java
    ├── GIFExporter.java
    ├── ImageCompressor.java
    ├── Main.java
    ├── Node.java
    └── QuadTree.java
|
└── test/
    ├── cats.jpg
    └── deer.png
```

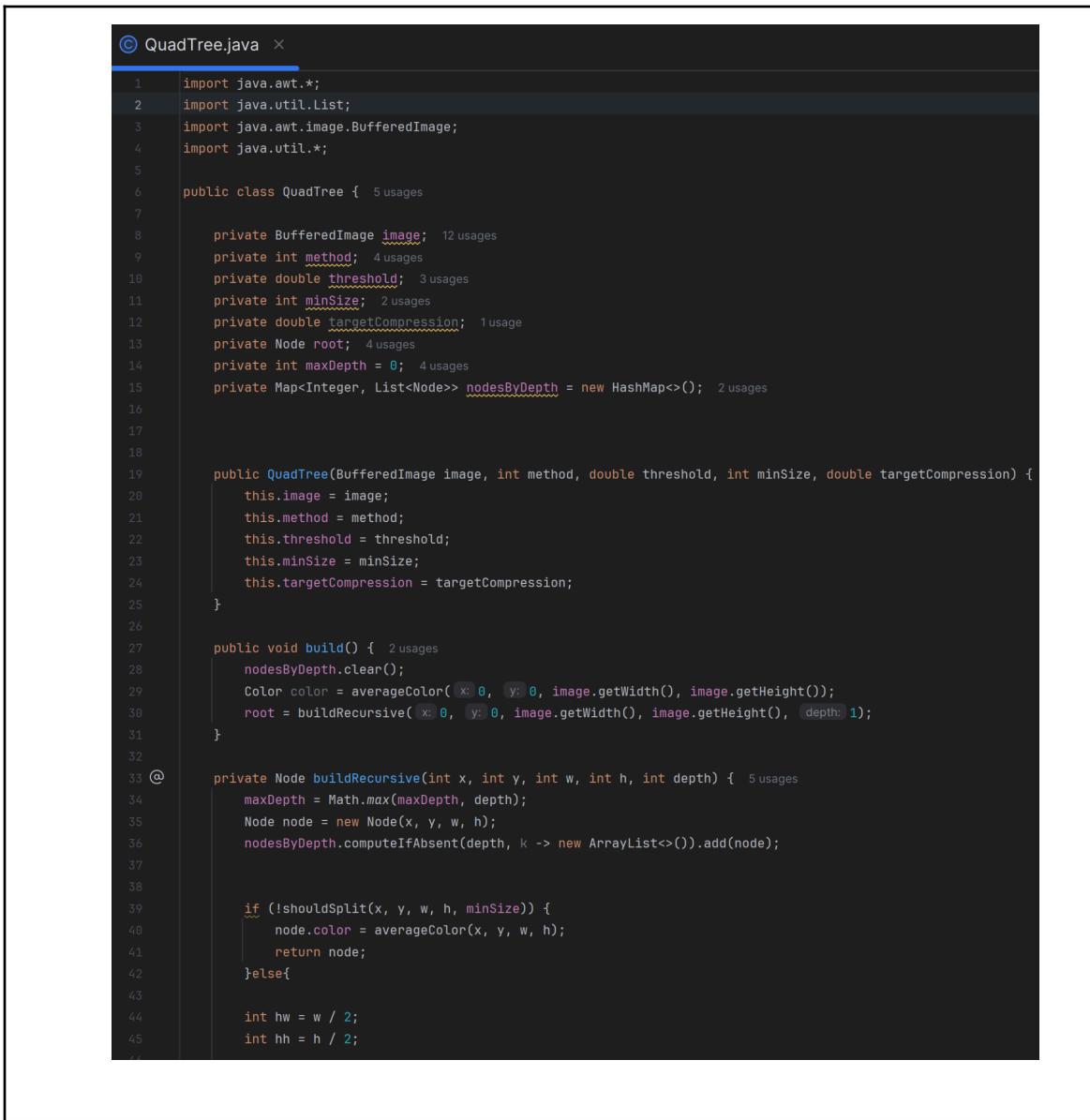
```
|   └── IceCream.jpeg  
|   └── iruma-kun.png  
|   └── karina.png  
|   └── kucing.jpg  
|   └── Landscape.jpeg  
  
|── bin/  
  
|── doc/  
|   └── Tucil2_K3_13523143_13523157.pdf  
  
└── README.md
```

4.3. Struktur Data Program

4.3.1. QuadTree.java

QuadTree.java merupakan file kelas utama yang merepresentasikan struktur pohon Quadtree. File ini bertanggung jawab atas pembentukan pohon, rekonstruksi gambar, dan pembuatan frame GIF berdasarkan kedalaman pohon.

A. Kode Program QuadTree.java



The screenshot shows a code editor window with the title bar "QuadTree.java". The code is a Java class definition for "QuadTree". The code includes imports for java.awt.* and java.util.List, and defines private fields for image, method, threshold, minSize, targetCompression, root, maxDepth, and nodesByDepth. It contains a constructor, a build() method, and a buildRecursive() method. The buildRecursive() method includes logic for splitting nodes based on size and color averaging.

```
① import java.awt.*;
② import java.util.List;
③ import java.awt.image.BufferedImage;
④ import java.util.*;
⑤
⑥ public class QuadTree { 5 usages
⑦
⑧     private BufferedImage image; 12 usages
⑨     private int method; 4 usages
⑩     private double threshold; 3 usages
⑪     private int minSize; 2 usages
⑫     private double targetCompression; 1 usage
⑬     private Node root; 4 usages
⑭     private int maxDepth = 0; 4 usages
⑮     private Map<Integer, List<Node>> nodesByDepth = new HashMap<>(); 2 usages
⑯
⑰
⑱
⑲     public QuadTree(BufferedImage image, int method, double threshold, int minSize, double targetCompression) {
⑳         this.image = image;
⑳         this.method = method;
⑳         this.threshold = threshold;
⑳         this.minSize = minSize;
⑳         this.targetCompression = targetCompression;
⑳     }
⑳
⑳     public void build() { 2 usages
⑳         nodesByDepth.clear();
⑳         Color color = averageColor( x: 0, y: 0, image.getWidth(), image.getHeight());
⑳         root = buildRecursive( x: 0, y: 0, image.getWidth(), image.getHeight(), depth: 1);
⑳     }
⑳
⑳     @
⑳     private Node buildRecursive(int x, int y, int w, int h, int depth) { 5 usages
⑳         maxDepth = Math.max(maxDepth, depth);
⑳         Node node = new Node(x, y, w, h);
⑳         nodesByDepth.computeIfAbsent(depth, k -> new ArrayList<>()).add(node);
⑳
⑳         if (!shouldSplit(x, y, w, h, minSize)) {
⑳             node.color = averageColor(x, y, w, h);
⑳             return node;
⑳         } else{
⑳
⑳             int hw = w / 2;
⑳             int hh = h / 2;
```

```

        node.children = new Node[4];
        node.children[0] = buildRecursive(x, y, hw, hh, depth: depth + 1);
        node.children[1] = buildRecursive(x: x + hw, y, w: w - hw, hh, depth: depth + 1);
        node.children[2] = buildRecursive(x, y: y + hh, hw, h: h - hh, depth: depth + 1);
        node.children[3] = buildRecursive(x: x + hw, y: y + hh, w: w - hw, h: h - hh, depth: depth + 1);

        return node;
    }

    private boolean shouldSplit(int x, int y, int w, int h, int minSize) { 1 usage
        if ((w*h/4) < minSize) return false;
        if (method == 5){
            return ErrorCalculator.compute(image, x, y, w, h, method) < threshold;
        }
        else{
            return ErrorCalculator.compute(image, x, y, w, h, method) > threshold;
        }
    }

    public Color averageColor(int x, int y, int w, int h) { 3 usages
        long r = 0, g = 0, b = 0;
        int total = w * h;
        for (int i = x; i < x + w; i++) {
            for (int j = y; j < y + h; j++) {
                Color c = new Color(image.getRGB(i, j));
                r += c.getRed();
                g += c.getGreen();
                b += c.getBlue();
            }
        }
        return new Color((int)(r / total), (int)(g / total), (int)(b / total));
    }

    public BufferedImage reconstructImage() { 2 usages
        BufferedImage output = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
        drawNode(output, root);
        return output;
    }
}

```

```

private void drawNode(BufferedImage img, Node node) { 2 usages
    if (node.isLeaf()) {
        Graphics2D g = img.createGraphics();
        g.setColor(node.color);
        g.fillRect(node.x, node.y, node.width, node.height);
        g.dispose();
    } else {
        for (Node child : node.children) {
            drawNode(img, child);
        }
    }
}

public List<BufferedImage> generateGIFFrames() { 1 usage
    List<BufferedImage> frames = new ArrayList<>();
    for (int d = 1; d <= maxDepth; d++) {
        BufferedImage frame = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
        reconstructQuadTreeForGIF(frame, currentDepth: 0, d);
        frames.add(frame);
    }
    return frames;
}

public void reconstructQuadTreeForGIF(BufferedImage output, int currentDepth, int depthLimit) { 1 usage
    reconstructNodeForGIF(root, output, currentDepth, depthLimit);
}

private void reconstructNodeForGIF(Node node, BufferedImage output, int currentDepth, int depthLimit) { 2 usages
    if (node.isLeaf() || currentDepth >= depthLimit) {
        if (node.color == null) {
            node.color = averageColor(node.x, node.y, node.width, node.height);
        }
        for (int y = node.y; y < node.y + node.height; y++) {
            for (int x = node.x; x < node.x + node.width; x++) {
                output.setRGB(x, y, node.color.getRGB());
            }
        }
    }
    return;
}

```

```

private void reconstructNodeForGIF(Node node, BufferedImage output, int currentDepth, int depthLimit) { 2 usages
    if (node.isLeaf() || currentDepth >= depthLimit) {
        if (node.color == null) {
            node.color = averageColor(node.x, node.y, node.width, node.height);
        }
        for (int y = node.y; y < node.y + node.height; y++) {
            for (int x = node.x; x < node.x + node.width; x++) {
                output.setRGB(x, y, node.color.getRGB());
            }
        }
        return;
    }

    for (Node child : node.children) {
        reconstructNodeForGIF(child, output, currentDepth: currentDepth + 1, depthLimit);
    }
}

public int countTotalNodes() { return countNodes(root); }

private int countNodes(Node node) { 2 usages
    if (node == null) return 0;
    int count = 1;
    if (!node.isLeaf()) {
        for (Node child : node.children) count += countNodes(child);
    }
    return count;
}

public int getMaxDepth() { return maxDepth; }
}

```

B. Atribut QuadTree

Nama Atribut	Tipe Data	Keterangan
image	BufferedImage	Objek gambar input yang akan dikompresi
method	int	Metode penghitungan error yang digunakan (1–5)
threshold	double	Ambang batas error untuk pembagian blok
minSize	int	Ukuran minimum blok yang boleh dibagi
targetCompression	double	Target persentase kompresi (0.0–1.0)

root	Node	Akar dari pohon Quadtree
maxDepth	int	Kedalaman maksimum pohon
nodesByDepth	Map<Integer, List<Node>>	Daftar node yang dikelompokkan berdasarkan tingkat kedalaman

C. Metode QuadTree

Nama Metode	Keterangan
QuadTree(...)	Konstruktor untuk menginisialisasi atribut Quadtree.
build()	Digunakan untuk memulai proses pembentukan pohon dari gambar
buildRecursive(...)	Rekursi untuk membagi blok menjadi simpul-simpul pohon
shouldSplit(...)	Digunakan untuk mengecek apakah suatu blok harus dibagi berdasarkan threshold & ukuran
averageColor(...)	Digunakan untuk menghitung rata-rata warna blok
reconstructImage()	Digunakan untuk menghasilkan gambar hasil kompresi berdasarkan pohon
drawNode(...)	Digunakan untuk menggambar node leaf ke dalam gambar
generateGIFFrames()	Digunakan untuk menghasilkan frame untuk visualisasi proses Quadtree
reconstructQuadTreeForGIF(...)	Digunakan untuk membangun ulang frame berdasarkan kedalaman tertentu
reconstructNodeForGIF(...)	Prosedur rekursif untuk membentuk setiap blok pada kedalaman tertentu.

countTotalNodes()	Digunakan untuk menghitung total jumlah simpul dalam pohon
countNodes(...)	Fungsi bantu rekursif untuk menghitung simpul
getMaxDepth()	Digunakan untuk mengembalikan kedalaman maksimum pohon

4.3.2. Node.java

Node.java merupakan file berisi struktur simpul (*node*) dalam pohon Quadtree. File ini berfungsi untuk merepresentasikan satu blok gambar dan menyimpan data koordinat, ukuran, warna rata-rata, serta referensi ke *children nodes*.

A. Kode Program Node.java

```
import java.awt.Color;

public class Node { 13 usages
    public int x, y, width, height; 5 usages
    public Color color; 5 usages
    public Node[] children; 9 usages

    public Node(int x, int y, int width, int height) { 1 usage
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
    }

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

B. Atribut Node

Nama Atribut	Tipe Data	Keterangan
x	int	Koordinat X dari blok
y	int	Koordinat Y dari blok
width	int	Lebar blok
height	int	Tinggi blok
color	Color	Warna rata-rata dari blok
children	Node[]	Anak simpul (maksimal 4), null jika leaf

C. Metode QuadTree

Nama Metode	Keterangan
Node(...)	Konstruktor node dengan posisi dan ukuran
isLeaf()	Untuk mengecek apakah node adalah leaf

4.3.3. ImageCompressor.java

ImageCompressor.java merupakan yang berfungsi untuk menangani proses kompresi dari awal hingga akhir. File ini bertugas untuk membaca input, menentukan threshold, membentuk Quadtree, dan menyimpan hasil gambar serta informasi statistik.

A. Kode Program ImageCompressor.java

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

public class ImageCompressor { 1usage
    public static void run(String inputPath, int method, double threshold, int minBlockSize, 1usage
                           double targetCompression, String outputPath, String gifPath) {
        try {
            long startTime = System.nanoTime();
            BufferedImage image = ImageIO.read(new File(inputPath));

            QuadTree tree;
            GIFExporter gif = (gifPath != null && !gifPath.trim().isEmpty()) ? new GIFExporter(gifPath, delayMs: 500) : null;

            if (targetCompression > 0.0 && targetCompression <= 1.0) { //Jika target kompresi di set
                double low, high, bestThresh;

                switch (method) {
                    case 1 -> { //Variance
                        low = 0.0;
                        high = 16384.0;
                    }
                    case 2 -> { //MAD
                        low = 0.0;
                        high = 255.0;
                    }
                    case 3 -> { //Max Pixel Difference
                        low = 0.0;
                        high = 255.0;
                    }
                    case 4 -> { //Entropy
                        low = 0.0;
                        high = 8.0;
                    }
                    case 5 -> { //SSIM
                        low = -1.0; //avoid error
                        high = 1.0;
                    }
                    default -> {
                        low = 0.0;
                        high = 10000.0;
                    }
                }

                bestThresh = threshold;
            }
        }
    }
}
```

```
    double epsilon = 0.1;
    int maxIterations = 30;

    File inputFile = new File(inputPath);
    long originalSize = inputFile.length();

    for (int i = 0; i < maxIterations; i++) {
        double mid = (low + high) / 2.0;
        QuadTree temp = new QuadTree(image, method, mid, minBlockSize, targetCompression);
        temp.build();

        BufferedImage output = temp.reconstructImage();

        File tempFile = File.createTempFile( prefix: "compressed_", suffix: ".png");
        ImageIO.write(output, formatName: "png", tempFile);

        long compressedSize = tempFile.length();
        double compressionRatio = 1.0 - ((double) compressedSize / originalSize);

        tempFile.delete();

        if (method == 5) {
            if (compressionRatio > targetCompression) {
                low = mid;
                bestThresh = mid;
            } else {
                high = mid;
            }
        }
        else {
            if (compressionRatio < targetCompression) {
                low = mid;
            } else {
                high = mid;
                bestThresh = mid;
            }
        }
    }
}
```

```

        if (Math.abs(high - low) < epsilon) {
            break;
        }
    }

    System.out.printf("Threshold terbaik ditemukan: %.4f untuk target kompresi %.2f%\n",
                      bestThresh, targetCompression * 100.0);

    tree = new QuadTree(image, method, bestThresh, minBlockSize, targetCompression);
} else {
    // Mode manual
    tree = new QuadTree(image, method, threshold, minBlockSize, targetCompression);
}

tree.build();

if (gif != null) {
    for (BufferedImage frame : tree.generateGIFFrames()) {
        gif.addFrame(frame);
    }
    gif.close();
}

BufferedImage output = tree.reconstructImage();
ImageIO.write(output, formatName: "png", new File(outputPath));

long endTime = System.nanoTime();
double duration = (endTime - startTime) / 1e6; //ms

File inputFile = new File(inputPath);
File outputFile = new File(outputPath);

long originalSize = inputFile.length();
long compressedSize = outputFile.length();

double compressionPercent = 100.0 * (1 - (double) compressedSize / originalSize);

```

```

        System.out.println("HASIL: \n");
        System.out.printf("Waktu eksekusi: %.2f ms\n", duration);
        System.out.println("Ukuran gambar sebelum: " + originalSize);
        System.out.println("Ukuran gambar sesudah: " + compressedSize);
        System.out.printf("Percentase kompresi: %.2f%\n", compressionPercent);
        System.out.println("Kedalaman pohon: " + tree.getMaxDepth());
        System.out.println("Jumlah simpul: " + tree.countTotalNodes());

        System.out.printf("\nSuccess! Output disimpan di: %s\n", outputPath);

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

```

B. Atribut ImageCompressor

File ini tidak memiliki atribut.

C. Metode ImageCompressor

Nama Metode	Keterangan
run(...)	Prosedur utama yang menjalankan seluruh proses kompresi dari input ke output

4.3.4. ErrorCalculator.java

ErrorCalculator.java merupakan file yang berfungsi untuk menyediakan berbagai metode penghitungan error, yaitu *Variance*, *MAD*, *MaxDiff*, *Entropy*, dan *SSIM* yang digunakan sebagai dasar keputusan pembagian blok dalam Quadtree.

A. Kode Program ErrorCalculator.java

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

public class ErrorCalculator { 2 usages
    public static double compute(BufferedImage img, int x, int y, int w, int h, int method) { 2 usages
        return switch (method) {
            case 1 -> variance(img, x, y, w, h);
            case 2 -> mad(img, x, y, w, h);
            case 3 -> maxDiff(img, x, y, w, h);
            case 4 -> entropy(img, x, y, w, h);
            case 5 -> structuralSimilarity(img, x, y, w, h);
            default -> 0;
        };
    }

    //variance
    public static double variance(BufferedImage img, int x, int y, int width, int height) { 1 usage
        double[] sum = {0, 0, 0};
        double[] sumSq = {0, 0, 0};
        int n = width * height;

        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                Color c = new Color(img.getRGB(i, j));
                sum[0] += c.getRed();
                sum[1] += c.getGreen();
                sum[2] += c.getBlue();
                sumSq[0] += c.getRed() * c.getRed();
                sumSq[1] += c.getGreen() * c.getGreen();
                sumSq[2] += c.getBlue() * c.getBlue();
            }
        }

        double varR = sumSq[0] / n - Math.pow(sum[0] / n, 2);
        double varG = sumSq[1] / n - Math.pow(sum[1] / n, 2);
        double varB = sumSq[2] / n - Math.pow(sum[2] / n, 2);

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

```
//MAD
public static double mad(BufferedImage img, int x, int y, int width, int height) { 1 usage
    double[] mean = {0, 0, 0};
    int n = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color c = new Color(img.getRGB(i, j));
            mean[0] += c.getRed();
            mean[1] += c.getGreen();
            mean[2] += c.getBlue();
        }
    }

    mean[0] /= n;
    mean[1] /= n;
    mean[2] /= n;

    double[] total = {0, 0, 0};
    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color c = new Color(img.getRGB(i, j));
            total[0] += Math.abs(c.getRed() - mean[0]);
            total[1] += Math.abs(c.getGreen() - mean[1]);
            total[2] += Math.abs(c.getBlue() - mean[2]);
        }
    }

    return (total[0] + total[1] + total[2]) / (3.0 * n);
}
```

```

//max pixel difference
public static double maxDiff(BufferedImage img, int x, int y, int width, int height) { 1usage
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color c = new Color(img.getRGB(i, j));
            int r = c.getRed(), g = c.getGreen(), b = c.getBlue();
            if (r < minR) minR = r;
            if (r > maxR) maxR = r;
            if (g < minG) minG = g;
            if (g > maxG) maxG = g;
            if (b < minB) minB = b;
            if (b > maxB) maxB = b;
        }
    }

    return ((maxR - minR) + (maxG - minG) + (maxB - minB)) / 3.0;
}

//entropy
public static double entropy(BufferedImage img, int x, int y, int width, int height) { 1usage
    int[] histogram = new int[256];
    int n = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color c = new Color(img.getRGB(i, j));
            int avg = (c.getRed() + c.getGreen() + c.getBlue()) / 3;
            histogram[avg]++;
        }
    }

    double entropy = 0.0;
    for (int count : histogram) {
        if (count > 0) {
            double p = (double) count / n;
            entropy -= p * (Math.log(p) / Math.log(2));
        }
    }

    return entropy;
}

```

```

//ssim
public static double structuralSimilarity(BufferedImage image, int x, int y, int width, int height) { 1 usage
    long r = 0, g = 0, b = 0;
    int total = width * height;
    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 avgR = (int)(r / total);
    int avgG = (int)(g / total);
    int avgB = (int)(b / total);

    double ssimR = computeSSIM(image, avgR, x, y, width, height, channel: 'r');
    double ssimG = computeSSIM(image, avgG, x, y, width, height, channel: 'g');
    double ssimB = computeSSIM(image, avgB, x, y, width, height, channel: 'b');

    return 0.2989 * ssimR + 0.5870 * ssimG + 0.1140 * ssimB;
}

private static double computeSSIM(BufferedImage image, int meanY, int x, int y, int width, int height, char channel) { 3 usages
    final double C1 = 6.5025;
    final double C2 = 58.5225;

    double sumX = 0, sumX2 = 0, sumXY = 0;
    int N = width * height;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            Color c = new Color(image.getRGB(i, j));
            int valX = switch (channel) {
                case 'r' -> c.getRed();
                case 'g' -> c.getGreen();
                case 'b' -> c.getBlue();
                default -> -1;
            };
            sumX += valX;
            sumX2 += valX * valX;
            sumXY += valX * meanY;
        }
    }

    double meanX = sumX / N;
    double varianceX = sumX2 / N - meanX * meanX;
    double varianceY = 0;
    double covariance = sumXY / N - meanX * meanY;

    double numerator = (2 * meanX * meanY + C1) * (2 * covariance + C2);
    double denominator = (meanX * meanX + meanY * meanY + C1) * (varianceX + varianceY + C2);

    return numerator / denominator;
}
}

```

B. Atribut ErrorCalculator

File ini tidak memiliki atribut.

C. Metode ErrorCalculator

Nama Metode	Keterangan
<code>compute(...)</code>	Digunakan untuk memilih metode error yang sesuai berdasarkan input dan menghitung nilainya.
<code>variance(...)</code>	Digunakan untuk menghitung variansi nilai warna RGB dalam blok.
<code>mad(...)</code>	Digunakan untuk menghitung Mean Absolute Deviation (MAD) dalam blok.
<code>maxDiff(...)</code>	Digunakan untuk menghitung selisih nilai maksimum dan minimum warna dalam blok.
<code>entropy(...)</code>	Digunakan untuk menghitung entropi distribusi warna rata-rata dari blok.
<code>structuralSimilarity(...)</code>	Digunakan untuk menghitung nilai <i>Structural Similarity Index</i> (SSIM) antar blok.
<code>computeSSIM(...)</code>	Metode internal bantu menghitung SSIM untuk satu kanal warna

4.3.5. GIFExporter.java

GIFExporter.java merupakan file yang berfungsi untuk menghasilkan file animasi (gif). File ini juga secara keseluruhan digunakan untuk visualisasi proses kompresi Quadtree.

A. Kode Program GIFExporter.java

```
import javax.imageio.*;
import javax.imageio.stream.ImageOutputStream;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.ImageTypeSpecifier;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

public class GIFExporter { 2 usages
    private ImageWriter writer; 6 usages
    private IIOMetadata metadata; 4 usages
    private ImageOutputStream output; 3 usages

    public GIFExporter(String path, int delayMs) throws IOException { 1 usage
        writer = ImageIO.getImageWritersBySuffix(fileSuffix: "gif").next();
        output = ImageIO.createImageOutputStream(new File(path));
        writer.setOutput(output);
        writer.prepareWriteSequence(streamMetadata: null);

        ImageTypeSpecifier type = ImageTypeSpecifier.createFromBufferedImageType(BufferedImage.TYPE_INT_RGB);
        metadata = writer.getDefaultImageMetadata(type, param: null);
        String format = metadata.getNativeMetadataFormatName();

        IIOMetadataNode root = new IIOMetadataNode(format);
        IIOMetadataNode gce = new IIOMetadataNode(nodeName: "GraphicControlExtension");
        gce.setAttribute(name: "disposalMethod", value: "none");
        gce.setAttribute(name: "userInputFlag", value: "FALSE");
        gce.setAttribute(name: "transparentColorFlag", value: "FALSE");
        gce.setAttribute(name: "delayTime", Integer.toString(i: delayMs / 10));
        gce.setAttribute(name: "transparentColorIndex", value: "0");
        root.appendChild(gce);

        IIOMetadataNode appExtensions = new IIOMetadataNode(nodeName: "ApplicationExtensions");
        IIOMetadataNode appExtension = new IIOMetadataNode(nodeName: "ApplicationExtension");
        appExtension.setAttribute(name: "applicationID", value: "NETSCAPE");
        appExtension.setAttribute(name: "authenticationCode", value: "2.0");
        appExtension.setUserObject(new byte[]{0x1, 0x0, 0x0});
        appExtensions.appendChild(appExtension);
        root.appendChild(appExtensions);

        metadata.mergeTree(format, root);
    }

    public void addFrame(BufferedImage image) throws IOException { 1 usage
        IIIOImage frame = new IIIOImage(image, thumbnails: null, metadata);
        writer.writeToSequence(frame, param: null);
    }
}
```

```

    public void close() throws IOException { 1 usage
        writer.endWriteSequence();
        output.close();
    }

}

```

B. Atribut GIFExporter

Nama Atribut	Tipe Data	Keterangan
writer	ImageWriter	Digunakan untuk menulis format gambar .gif
metadata	IIMetadata	Metadata gambar yang digunakan dalam setiap frame GIF
output	ImageOutputStream	Stream output untuk file GIF yang akan dihasilkan

C. Metode GIFExporter

Nama Metode	Keterangan
GIFExporter(...)	Konstruktor yang menginisialisasi writer dan metadata untuk GIF
addFrame(...)	Digunakan untuk menambahkan satu frame BufferedImage ke sequence GIF
close()	Digunakan untuk menutup writer dan stream output

4.3.6. Main.java

Main.java merupakan file yang berfungsi sebagai entry point program. File ini bertugas untuk menangani input dari pengguna melalui terminal. Secara keseluruhan, file ini bertugas dalam mengatur alur interaksi awal antara pengguna dan proses utama program.

A. Kode Main.java

```
import java.io.File;
import java.util.Scanner;

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

        String inputPath;
        while (true) {
            System.out.print("Masukkan path gambar input (.png/.jpg/.jpeg): ");
            inputPath = sc.nextLine().trim();
            File inputFile = new File(inputPath);
            if (inputFile.exists() && inputFile.isFile()
                && (inputPath.endsWith(".png") || inputPath.endsWith(".jpg") || inputPath.endsWith(".jpeg"))) {
                break;
            }
            System.out.println("Error: File tidak ditemukan atau format tidak sesuai!");
        }

        int errorMethod;
        while (true) {
            System.out.print("Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): ");
            if (sc.hasNextInt()) {
                errorMethod = sc.nextInt();
                if (errorMethod >= 1 && errorMethod <= 5) {
                    break;
                }
            }
            System.out.println("Error: Pilihan tidak valid!");
            sc.nextLine();
        }
    }
}
```

```

        System.out.print("Masukkan nilai threshold ");
        switch (errorMethod) {
            case 1 -> System.out.print("Variance (ideal range [0..16384]): ");
            case 2 -> System.out.print("Mean Absolute Deviation (ideal range [0..255]): ");
            case 3 -> System.out.print("Max Pixel Difference (ideal range [0..255]): ");
            case 4 -> System.out.print("Entropy (ideal range [0..8]): ");
            case 5 -> System.out.print("Structural Similarity Index (ideal range [0..1]): ");
        }
        double threshold = sc.nextDouble();

        int minBlockSize;
        while (true) {
            System.out.print("Masukkan ukuran blok minimum (>=1): ");
            if (sc.hasNextInt()) {
                minBlockSize = sc.nextInt();
                if (minBlockSize > 0) break;
            }
            System.out.println("Error: Ukuran blok minimum harus >= 1!");
            sc.nextLine();
        }

        double targetCompression;
        while (true) {
            System.out.print("Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): ");
            if (sc.hasNextDouble()) {
                targetCompression = sc.nextDouble();
                if (targetCompression >= 0 && targetCompression <= 1) break;
            }
            System.out.println("Error: Masukkan angka antara 0.0 dan 1.0!");
            sc.nextLine();
        }
        sc.nextLine();

        String outputPath;
        while (true) {
            System.out.print("Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): ");
            outputPath = sc.nextLine().trim();
            if (outputPath.endsWith(".png") || outputPath.endsWith(".jpg") || outputPath.endsWith(".jpeg")) {
                break;
            }
            System.out.println("Error: Path harus berakhiran (.png/.jpg/.jpeg!)");
        }
    }
}

```

```

String gifPath;
while (true) {
    System.out.print("Masukkan path GIF (kosongkan jika tidak perlu): ");
    gifPath = sc.nextLine().trim();
    if (gifPath.isEmpty() || gifPath.endsWith(".gif")) {
        break;
    }
    System.out.println("Error: Path harus berakhiran .gif atau kosong!");
}

sc.close();

ImageCompressor.run(inputPath, errorMethod, threshold, minBlockSize, targetCompression, outputPath, gifPath);
}

```

B. Atribut Main

File ini tidak memiliki atribut.

C. Metode Main

Nama Metode	Keterangan
main(...)	Fungsi utama public static void main yang bertugas untuk menerima input dan menjalankan kompresi.

BAB V HASIL DAN ANALISIS PENGUJIAN

5.1. Test Case 1 (*Variance*)

Input	
<pre>Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\tucil Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1 Masukkan nilai threshold Variance (ideal range [0..16384]): 200 Masukkan ukuran blok minimum (>=1): 8 Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0 Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\tucil Masukkan path GIF (kosongkan jika tidak perlu): C:\Users\Amira\Documents\STIMA\tucil</pre>	
Output	
<pre>HASIL: Waktu eksekusi: 3479,21 ms Ukuran gambar sebelum: 413773 Ukuran gambar sesudah: 233503 Persentase kompresi: 43,57% Kedalaman pohon: 10 Jumlah simpul: 47261 Success! Output disimpan di: C:\Users\Amira\Documents\STIMA\tucil</pre>	

5.2. Test Case 2 (*MAD*)

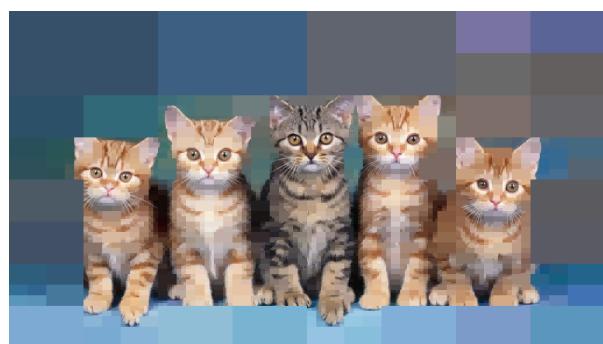
Input

```
Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tuci  
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 2  
Masukkan nilai threshold Mean Absolute Deviation (ideal range [0..127.5]): 16  
Masukkan ukuran blok minimum (>=1): 3  
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0  
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA  
Masukkan path GIF (kosongkan jika tidak perlu):  
MASUKKAN
```



Output

```
HASIL:  
  
Waktu eksekusi: 980,85 ms  
Ukuran gambar sebelum: 413773  
Ukuran gambar sesudah: 181389  
Persentase kompresi: 56,16%  
Kedalaman pohon: 11  
Jumlah simpul: 30653  
  
Success! Output disimpan di: C:\Users\Amira\Documents\STIMA\Tuci
```



5.3. Test Case 3 (*Max pixel difference*)

Input

```
Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tuci  
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 3  
Masukkan nilai threshold Max Pixel Difference (ideal range [0..255]): 15  
Masukkan ukuran blok minimum (>=1): 15  
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0  
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA  
Masukkan path GIF (kosongkan jika tidak perlu):
```



Output

HASIL:

```
Waktu eksekusi: 980,85 ms
Ukuran gambar sebelum: 413773
Ukuran gambar sesudah: 181389
Persentase kompresi: 56,16%
Kedalaman pohon: 11
Jumlah simpul: 30653

Success! Output disimpan di: C:\Users\Amira
```



5.4. Test Case 4 (*Entropy*)

Input

```
Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tuc
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 4
Masukkan nilai threshold Entropy (ideal range [0..8]): 6
Masukkan ukuran blok minimum (>=1): 16
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STI
Masukkan path GIF (kosongkan jika tidak perlu):
```

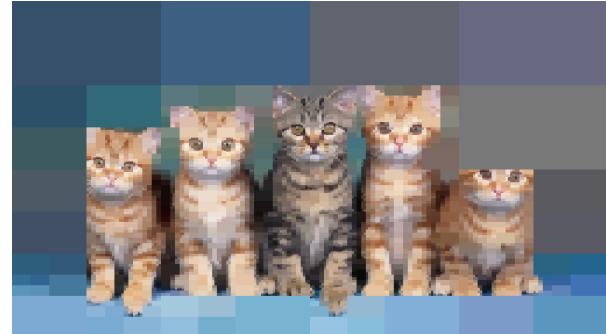


Output

HASIL:

```
Waktu eksekusi: 699,07 ms
Ukuran gambar sebelum: 413773
Ukuran gambar sesudah: 135409
Persentase kompresi: 67,27%
Kedalaman pohon: 9
Jumlah simpul: 8441

Success! Output disimpan di: C:\Users
```



5.5. Test Case 5 (*SSIM*)

Input

```
Masukkan path gambar input (.png/.jpeg): C:\Users\Amira\Documents\STIMA\Tucil2_1  
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 5  
Masukkan nilai threshold Structural Similarity Index (ideal range [0..1]): 0,8  
Masukkan ukuran blok minimum (>=1): 128  
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0,0 hingga 1,0): 0  
Masukkan path untuk gambar hasil (.png/.jpeg): C:\Users\Amira\Documents\STIMA\Tucil2_1\hasil  
Masukkan path GIF (kosongkan jika tidak perlu): C:\Users\Amira\Documents\STIMA\Tucil2_1\hasil
```



Output

```
Masukkan path GIF (kosongkan jika tidak perlu):  
HASIL:  
  
Waktu eksekusi: 3679,88 ms  
Ukuran gambar sebelum: 413773  
Ukuran gambar sesudah: 173837  
Persentase kompresi: 57,99%  
Kedalaman pohon: 7  
Jumlah simpul: 4597  
  
Success! Output disimpan di: C:\Users\Amira\Documents\STIMA\Tucil2_1\hasil
```



5.6. Test Case 6 (*Variance with Target Compression*)

Input

```
Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tucil2  
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1  
Masukkan nilai threshold Variance (ideal range [0..16384]): 128  
Masukkan ukuran blok minimum (>=1): 8  
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0,4  
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tucil2\hasil.png  
Masukkan path GIF (kosongkan jika tidak perlu):  
Threshold terbaik ditemukan: 173,7500 untuk target kompresi 40,00%
```



Output

HASIL:

```
Waktu eksekusi: 10499,46 ms  
Ukuran gambar sebelum: 413773  
Ukuran gambar sesudah: 248254  
Percentase kompresi: 40,00%  
Kedalaman pohon: 10  
Jumlah simpul: 52549  
  
Success! Output disimpan di: C:\Users\Amira\Documents\STIMA\Tucil2\hasil.png
```



5.7. Test Case 7 (*SSIM with Target Compression*)

Input

```

Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tugas\Pengembangan Aplikasi\Kompresi\kucing1.jpg
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
Masukkan nilai threshold Structural Similarity Index (ideal range [0..1]): 0.1
Masukkan ukuran blok minimum (>=1): 16
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0.35
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tugas\Pengembangan Aplikasi\Kompresi\kucing1_35.jpg
Masukkan path GIF (kosongkan jika tidak perlu):
Threshold terbaik ditemukan: 0,8750 untuk target kompresi 35,00%
HASIL:

```



Output

```

HASIL:

Waktu eksekusi: 7379,84 ms
Ukuran gambar sebelum: 413773
Ukuran gambar sesudah: 261788
Persentase kompresi: 36,73%
Kedalaman pohon: 9
Jumlah simpul: 56449

Success! Output disimpan di: C:\Use

```



5.8. Test Case 8 (*Variance with Target compression and different image*)

Input

```

Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tugas\Pengembangan Aplikasi\Kompresi\drumashun.jpg
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 1
Masukkan nilai threshold Variance (ideal range [0..16384]): 128
Masukkan ukuran blok minimum (>=1): 8
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0,4
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tugas\Pengembangan Aplikasi\Kompresi\drumashun_40.jpg
Masukkan path GIF (kosongkan jika tidak perlu):
Threshold terbaik ditemukan: 15,1250 untuk target kompresi 40,00%
HASIL:

```



Output

HASIL:

```
Waktu eksekusi: 5638,67 ms
Ukuran gambar sebelum: 881828
Ukuran gambar sesudah: 528745
Persentase kompresi: 40,04%
Kedalaman pohon: 10
Jumlah simpul: 163697

Success! Output disimpan di: C:\User
```



5.9. Test Case 9 (*Entropy with high quality image*)

Input

```
Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tu...
Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 4
Masukkan nilai threshold Entropy (ideal range [0..8]): 6
Masukkan ukuran blok minimum (>1): 16
Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0
Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\ST...
Masukkan path GIF (kosongkan jika tidak perlu):
MASUK...
```

**Output**

<pre> HASIL: Waktu eksekusi: 801,24 ms Ukuran gambar sebelum: 3238078 Ukuran gambar sesudah: 61010 Persentase kompresi: 98,19% Kedalaman pohon: 7 Jumlah simpul: 65 Success! Output disimpan di: C:\Users\Amira\Documents\STIMA\Tuc12_13523142_13523157\test\kerina_2.png </pre>	
---	--

5.10. Test Case 10 (*Entropy with low quality image*)

Input	
<pre> Masukkan path gambar input (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tuc12_13523142_13523157\test\kerina_2.jpg Pilih metode error (1: Variance, 2: MAD, 3: MaxDiff, 4: Entropy, 5: SSIM): 4 Masukkan nilai threshold Entropy (ideal range [0..8]): 6 Masukkan ukuran blok minimum (>1): 16 Masukkan target persentase kompresi (0 untuk nonaktifkan, 0.0 hingga 1.0): 0 Masukkan path untuk gambar hasil (.png/.jpg/.jpeg): C:\Users\Amira\Documents\STIMA\Tuc12_13523142_13523157\test\kerina_2.png Masukkan path GIF (kosongkan jika tidak perlu): </pre>	
Output	

<pre> HASIL: Waktu eksekusi: 621,58 ms Ukuran gambar sebelum: 126025 Ukuran gambar sesudah: 102285 Persentase kompresi: 18,84% Kedalaman pohon: 9 Jumlah simpul: 10169 Success! Output disimpan di: C:\User </pre>	
--	--

5.11. Analisis Test Case

Minimum ukuran blok mampu mempengaruhi hasil kompresi, yaitu semakin besar ukuran blok semakin tinggi persentase kompresi. Selain metode SSIM, semakin nilai threshold mendekati nilai maksimum threshold semakin tinggi persentase kompresi. Dalam metode SSIM terjadi hal yang sebaliknya Ketika mendekati nilai maksimum threshold semakin rendah persentase kompresi.

Dengan parameter target persentase kompresi, user bisa mendapatkan persentase input yang diinginkan. Target persentase kompresi ini menemukan threshold terbaik agar bisa mencapai target persentase kompresi. Namun, terdapat kondisi dimana saat melakukan target kompresi, hasil tidak sesuai dengan yang diinginkan. Hal ini kemungkinan terjadi karena terdapat area dengan warna yang homogen sehingga tidak bisa memecah warna Kembali karena perbedaan warna yang sangat kecil di area tersebut.

Ketika melakukan dengan test case yang sama dengan ukuran gambar yang berbeda (Test Case 4, 9, dan 10) menghasilkan persentase kompresi yang berbeda, kemungkinan hasil tersebut terjadi karena saat pembagian node area yang diambil akan lebih besar sehingga terdapat kemungkinan area tersebut homogen sehingga algoritma membentuk satu blok besar tanpa perlu membaginya lagi.

Ketika memasukkan input dengan ukuran minimum blok yang sangat kecil atau threshold yang kecil kemungkinan terjadi persentase kompresi dengan hasil negatif, yaitu ukuran hasil kompresi lebih besar dibandingkan ukuran awal. Terdapat kemungkinan yaitu karena dilakukan pembagian hingga ukuran minimum block yang sangat kecil membuat gambar lebih kompleks dari original menghasilkan ukuran gambar yang lebih besar atau saat kompresi ke dalam .png/.jpg/.jpeg.

BAB VI IMPLEMENTASI BONUS

Dalam program tugas kecil 2 mengenai kompresi gambar dengan metode Quadtree, kami berhasil mengimplementasikan *Structural Similarity Index (SSIM)* sebagai sebuah metode pengukuran error, target persentase kompresi dan pembuatan GIF proses kompresi. Berikut merupakan penjelasan pengimplementasian setiap bonus:

6.1. SSIM

6.1.1. Kode Pengimplementasian SSIM

```
//ssim
public static double structuralSimilarity(BufferedImage image, int x, int y, int width, int height) {
    long r = 0, g = 0, b = 0;
    int total = width * height;
    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 avgR = (int)(r / total);
    int avgG = (int)(g / total);
    int avgB = (int)(b / total);

    double ssimR = computeSSIM(image, avgR, x, y, width, height, channel: 'r');
    double ssimG = computeSSIM(image, avgG, x, y, width, height, channel: 'g');
    double ssimB = computeSSIM(image, avgB, x, y, width, height, channel: 'b');

    return 0.2989 * ssimR + 0.5870 * ssimG + 0.1140 * ssimB;
}
```

```

private static double computeSSIM(BufferedImage image, int meanY, int x, int y, int width, int height, char channel)
{
    final double C1 = 6.5025;
    final double C2 = 58.5225;

    double sumX = 0, sumX2 = 0, sumXY = 0;
    int N = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color c = new Color(image.getRGB(i, j));
            int valX = switch (channel) {
                case 'r' -> c.getRed();
                case 'g' -> c.getGreen();
                case 'b' -> c.getBlue();
                default -> -1;
            };

            sumX += valX;
            sumX2 += valX * valX;
            sumXY += valX * meanY;
        }
    }

    double meanX = sumX / N;
    double varianceX = sumX2 / N - meanX * meanX;
    double varianceY = 0;
    double covariance = sumXY / N - meanX * meanY;

    double numerator = (2 * meanX * meanY + C1) * (2 * covariance + C2);
    double denominator = (meanX * meanX + meanY * meanY + C1) * (varianceX + varianceY + C2);

    return numerator / denominator;
}

```

6.1.2. Penjelasan Kode SSIM

Pengimplementasian algoritma *SSIM* (*Structural Similarity Index*) pada program ini dilakukan dengan menggunakan rumus dibawah ini:

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

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Algoritma SSIM terdiri dari dua fungsi utama, yaitu `structuralSimilarity` dan `computeSSIM`. Fungsi ini bekerja untuk menghitung tingkat kemiripan struktural suatu blok gambar berdasarkan tiga macam warna, yaitu merah, hijau, dan biru. Fungsi `structuralSimilarity` bertugas untuk menerima input berupa objek gambar `BufferedImage`,

serta informasi posisi dan ukuran blok (`x, y, width, height`). Fungsi ini dimulai dengan pertama menghitung rata-rata nilai warna untuk masing-masing RGB. Perhitungan dilakukan dengan menjumlahkan seluruh nilai warna dari tiap piksel di dalam blok, kemudian setelah itu membaginya dengan jumlah total piksel untuk memperoleh nilai `avgR`, `avgG`, dan `avgB`.

Setelah nilai rata-rata warna diperoleh, fungsi akan memanggil fungsi `computeSSIM` sebanyak tiga kali dan mengirimkan rata-rata kanal terkait. Fungsi `computeSSIM` bertanggung jawab menghitung nilai SSIM dengan cara mengambil ulang seluruh piksel dalam blok dan menghitung tiga komponen penting, yaitu `meanX` (rata-rata kanal), `varianceX` (variansi kanal), dan `covariance` (hubungan kanal X terhadap kanal Y). Nilai ini kemudian akan dihitung menggunakan rumus statistik. Perhitungan ini mempertimbangkan kuadrat nilai piksel, serta interaksi antara nilai kanal dan rata-rata referensinya. Setelah komponen statistik dihitung, nilai SSIM akan ditentukan dengan menggunakan rumus standar dengan konstanta `C1` dan `C2` untuk menghindari terjadinya pembagian nol. Hasil SSIM tiap kanal yang berupa R, G, B akan kemudian digabungkan kembali di fungsi `structuralSimilarity` untuk menghasilkan satu nilai SSIM gabungan. Terakhir, nilai yang telah didapat kemudian akan dibandingkan dengan threshold untuk memutuskan apakah blok gambar cukup seragam atau masih perlu dipecah.

6.2. Target Kompresi

6.2.1. Kode Pengimplementasian Target Kompresi

```
if (targetCompression > 0.0 && targetCompression <= 1.0) { //Jika target kompresi di set
    double low, high, bestThresh;

    switch (method) {
        case 1 -> { //Variance
            low = 0.0;
            high = 16384.0;
        }
        case 2 -> { //MAD
            low = 0.0;
            high = 255.0;
        }
        case 3 -> { //Max Pixel Difference
            low = 0.0;
            high = 255.0;
        }
        case 4 -> { //Entropy
            low = 0.0;
            high = 8.0;
        }
        case 5 -> { //SSIM
            low = -1.0; //avoid error
            high = 1.0;
        }
        default -> {
            low = 0.0;
            high = 10000.0;
        }
    }
}
```

```
bestThresh = threshold;

double epsilon = 0.1;
int maxIterations = 30;

File inputFile = new File(inputPath);
long originalSize = inputFile.length();

for (int i = 0; i < maxIterations; i++) {
    double mid = (low + high) / 2.0;
    QuadTree temp = new QuadTree(image, method, mid, minBlockSize, targetCompression);
    temp.build();

    BufferedImage output = temp.reconstructImage();

    File tempFile = File.createTempFile( prefix: "compressed_", suffix: ".png");
    ImageIO.write(output, formatName: "png", tempFile);

    long compressedSize = tempFile.length();
    double compressionRatio = 1.0 - ((double) compressedSize / originalSize);

    tempFile.delete();
```

```

        if (method == 5) {
            if (compressionRatio > targetCompression) {
                low = mid;
                bestThresh = mid;
            } else {
                high = mid;
            }
        }
        else {
            if (compressionRatio < targetCompression) {
                low = mid;
            } else {
                high = mid;
                bestThresh = mid;
            }
        }

        if (Math.abs(high - low) < epsilon) {
            break;
        }
    }

    System.out.printf("Threshold terbaik ditemukan: %.4f untuk target kompresi %.2f%\n",
                      bestThresh, targetCompression * 100.0);

    tree = new QuadTree(image, method, bestThresh, minBlockSize, targetCompression);
}

```

6.2.2. Penjelasan Kode Target Kompresi

Fitur target kompresi diimplementasikan dengan menggunakan algoritma berbentuk binary search. Algoritma ini bertujuan untuk mencari sebuah nilai threshold optimal yang dapat menghasilkan rasio kompresi sedekat mungkin dengan target yang diinput oleh pengguna. Program ini dimulai dengan pertama-tama menentukan rentang awal threshold. Rentang threshold ini dibagi antara dua, yaitu `low` dan `high`. Rentang ini ditentukan berdasarkan metode error yang digunakan. Setelah itu, algoritma akan memulai untuk melakukan iterasi untuk mencoba mencari berapa nilai threshold yang berada di tengah-tengah rentang tersebut. Kemudian, program akan membangun sebuah Quadtree, lalu menghitung ukuran file hasil kompresi.

Setelah berhasil mendapatkan ukuran file hasil kompresi tersebut, rasio kompresi akan kemudian dibandingkan dengan target. Lalu, rentang threshold akan disesuaikan berdasarkan hasil perbandingan. Jika nilai kompresi dinilai oleh program sudah cukup mendekati target atau

batas iterasi telah tercapai, maka proses akan dihentikan dan threshold terbaik yang ditemukan akan digunakan untuk kompresi akhir pada gambar.

6.3 GIF Visualisasi

6.3.1. Kode Pengimplementasian GIF Visualisasi

```
public List<BufferedImage> generateGIFFrames() { 1 usage  ± nataliadesiany
    List<BufferedImage> frames = new ArrayList<>();
    for (int d = 1; d <= maxDepth; d++) {
        BufferedImage frame = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
        reconstructQuadTreeForGIF(frame, currentDepth: 0, d);
        frames.add(frame);
    }
    return frames;
}

public void reconstructQuadTreeForGIF(BufferedImage output, int currentDepth, int depthLimit) { 1 usage  ± nataliadesiany
    reconstructNodeForGIF(root, output, currentDepth, depthLimit);
}

private void reconstructNodeForGIF(Node node, BufferedImage output, int currentDepth, int depthLimit) { 2 usages  ± na
    if (node.isLeaf() || currentDepth >= depthLimit) {
        if (node.color == null) {
            node.color = averageColor(node.x, node.y, node.width, node.height);
        }
        for (int y = node.y; y < node.y + node.height; y++) {
            for (int x = node.x; x < node.x + node.width; x++) {
                output.setRGB(x, y, node.color.getRGB());
            }
        }
        return;
    }

    for (Node child : node.children) {
        reconstructNodeForGIF(child, output, currentDepth: currentDepth + 1, depthLimit);
    }
}
```

6.3.2. Penjelasan Kode GIF Visualisasi

Algoritma program untuk menampilkan proses pembuatan GIF visualisasi Quadtree dilakukan dengan membentuk frame pada setiap level kedalaman pohon dan menggabungkannya menjadi urutan frame untuk GIF. Fungsi utama yang digunakan dalam pengimplementasian pembuatan kode ini adalah `generateGIFFrames()`. Fungsi ini bertujuan untuk menghasilkan list berisi objek yang berisikan kumpulan `BufferedImage`. `BufferedImage` akan dibuat untuk setiap kedalaman dari pohon pada Quadtree. Proses ini dimulai dari kedalaman 1 hingga memperoleh kedalaman maksimum. Untuk setiap kedalaman `d`, fungsi akan membuat gambar kosong dan memanggil fungsi `reconstructQuadTreeForGIF(...)`. Fungsi tersebut bertujuan untuk mengisi `image` dengan menggunakan struktur pohon hingga kedalaman yang didapat `d`.

Selanjutnya, fungsi `reconstructQuadTreeForGIF()` akan memanggil fungsi lain, yaitu `reconstructNodeForGIF()`. Fungsi ini akan menggambar blok gambar jika simpul tersebut merupakan *leaf* atau jika kedalaman saat ini sudah mencapai batas yang ditentukan. Setiap blok digambar dengan menggunakan warna rata-rata simpul dan diisi ke area yang sesuai dalam objek `BufferedImage` melalui penggunaan `setRGB()`. Proses ini akan terus dilakukan untuk seluruh node hingga kedalaman tertentu, sehingga gambar hasil pada setiap frame menunjukkan tahapan pertumbuhan pohon Quadtree secara visual. Semua frame yang telah dibuat akan kemudian digunakan oleh `GIFExporter` untuk menghasilkan file animasi berupa GIF.

BAB VII KESIMPULAN DAN SARAN

7.1. Kesimpulan

Berdasarkan hasil implementasi dan pengujian pembuatan kompresi gambar dengan metode Quadtree yang telah dilakukan, dapat disimpulkan bahwa kompresi gambar menggunakan struktur data Quadtree yang dikombinasikan dengan strategi algoritma *divide and conquer* mampu menghasilkan kompresi gambar yang optimal dan efisien. Proses pembagian blok berdasarkan tingkat keseragaman warna terbukti efektif dalam mengurangi ukuran berkas gambar, dengan tetap menjaga kualitas visual agar tidak mengalami penurunan yang signifikan..

Penggunaan berbagai metode pengukuran error seperti *Variance*, *MAD*, *Max Pixel Difference*, dan *Entropy* juga memberikan fleksibilitas dalam menentukan pendekatan metode error yang paling sesuai dengan kebutuhan pengguna. Pemrosesan berbasis *divide and conquer* juga pembagian dan penyelesaian blok-blok gambar secara terstruktur. Proses ini dilakukan dengan membagi gambar menjadi bagian-bagian kecil dan menyelesaiakannya secara rekursif. Hal ini tidak hanya efektif dalam menyederhanakan proses kompresi, tetapi juga terbukti dapat meningkatkan efisiensi algoritma secara keseluruhan.

7.2. Saran

Berdasarkan pengalaman yang kami dapat selama pengerjaan proyek ini, kami menyarankan beberapa hal yang bisa ditingkatkan, yaitu sebagai berikut:

1. Optimalisasi performa algoritma

Proses rekursif dalam pembentukan kompresi gambar dengan metode Quadtree dapat dikembangkan lebih lanjut dengan melakukan pengujian lebih lanjut terhadap beragam jenis gambar dengan karakteristik visual yang berbeda. Melalui pengujian lebih lanjut dapat diidentifikasi bagian-bagian dari proses kompresi yang memakan banyak waktu, sehingga algoritma dapat dikembangkan menjadi semakin optimal.

2. Pengembangan antarmuka pengguna (GUI)

Untuk membuat pengalaman pengguna program menjadi lebih baik, program dapat kedepannya dikembangkan lebih lanjut dengan antarmuka grafis yang interaktif.

3. Mendukung format gambar yang lebih beragam

Program dapat dikembangkan agar mendukung format gambar lain, seperti BMP dan 16-bit file, agar program menjadi lebih fleksibel dan dapat digunakan dalam berbagai kebutuhan kompresi gambar.

LAMPIRAN

Tautan Repository GitHub:

https://github.com/amiraIzani/Tucil1_13523143.git

Tautan Drive Berisi Gambar Test Case:

<https://drive.google.com/drive/folders/19j57QYQyG5wstcxIwMMqSRQgMn4tT2L4?usp=sharing>

Tabel Checklist Spesifikasi Program:

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	✓	