

Tugas Kecil 2 IF2211 Strategi Algoritma
Kompresi Gambar Dengan Metode Quadtree



Oleh
Muhammad Farrel Wibowo (13523153)
Bryan P. Hutagalung (18222130)

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika – Institut Teknologi Bandung
Jl. Ganeshha 10, Bandung 40132

2025

Daftar Isi

Daftar Isi.....	1
Daftar Tabel.....	3
Daftar Gambar.....	4
Bab I Pendahuluan.....	5
1.1 Deskripsi Tugas.....	5
1.2 Ilustrasi Kasus.....	6
1.3 Parameter.....	8
Bab II Landasan Teori Algoritma Devide and Conquer.....	11
Bab III Metodologi Implementasi.....	14
3.1 Langkah-Langkah Devide and Conquer.....	14
3.1.1 Inisialisasi dan Persiapan Data.....	14
3.1.2 Divide – Membagi Gambar Menjadi Sub-Blok.....	14
3.1.3 Conquer – Menyelesaikan Setiap Sub-Blok Secara Rekursif.....	14
3.1.4 Combine – Menggabungkan Solusi Sub-Blok Menjadi Gambar Kompresi.....	15
3.1.5 Penghitungan Statistik dan Penyimpanan Hasil.....	15
3.1.6 Pseudocode.....	16
3.2 Source Program.....	17
3.2.1 Struktur File.....	17
3.2.2 CompressionStats.java.....	18
3.2.3 ImageCompressor.java.....	21
3.2.4 ErrorCalculator.java.....	26
3.2.5 ErrorMethod.java.....	33
3.2.6 Node.java.....	35
3.2.7 Quadtree.java.....	37
3.2.8 GifGenerator.java.....	48
3.2.9 ImageUtil.java.....	56
3.2.10 Main.java.....	58
3.3 Implementasi Bonus.....	67
3.3.1 Persentase Kompresi.....	67

3.3.2 Structural Similarity Index (SSIM).....	69
3.3.3 Gif Visualisasi Proses Pembentukan Quadtree.....	71
Bab IV Hasil dan Pembahasan.....	80
2.1 Hasil Eksperimen.....	80
2.1.1 Test Case 1.....	80
2.1.2 Test Case 2.....	82
2.1.3 Test Case 3.....	84
2.1.4 Test Case 4.....	86
2.1.5 Test Case 5.....	87
2.1.6 Test Case 6.....	89
2.1.7 Test Case 7.....	90
2.1.8 Test Case 8.....	92
2.1.9 Test Case 9.....	94
2.1.10 Test Case 10.....	95
2.2 Analisis dan Pembahasan.....	97
2.2.1 Kualitas Visual dan Efisiensi Kompresi.....	97
2.2.2 Pengaruh Parameter pada Hasil Kompresi.....	98
2.2.3 Visualisasi Proses Kompresi (GIF).....	99
2.2.4 Efektivitas Metode Divide and Conquer dalam Kompresi Quadtree.....	100
BAB V Penutup.....	101
3.1 Kesimpulan.....	101
3.2 Saran.....	101
Referensi.....	103
Lampiran.....	104

Daftar Tabel

Tabel 1. Metode Pengukuran Error.....	8
Tabel 2. Penilaian Kelengkapan Program.....	104

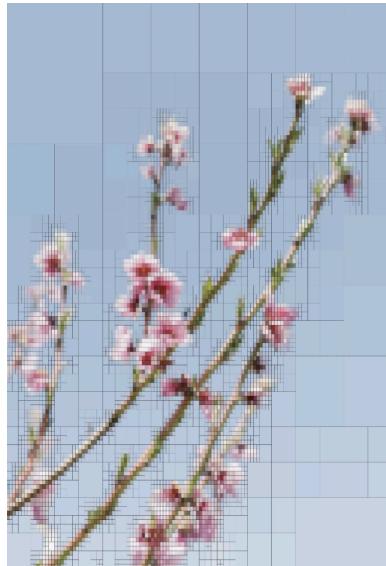
Daftar Gambar

Gambar 1. Quadtree dalam Kompresi Gambar	5
Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar	6
Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar	8
Gambar 4. Ilustrasi Algoritma Devide and Conquer	11

Bab I

Pendahuluan

1.1 Deskripsi Tugas



Gambar 1. Quadtree dalam Kompresi Gambar

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 1 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



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar

1.2 Ilustrasi Kasus

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

a. Metode perhitungan variansi

Pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada Tabel 1.

b. Threshold variansi

Nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.

c. Minimum block size

Ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai Tabel 1.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- a. Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- b. Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

5. Rekursi dan Penghentian

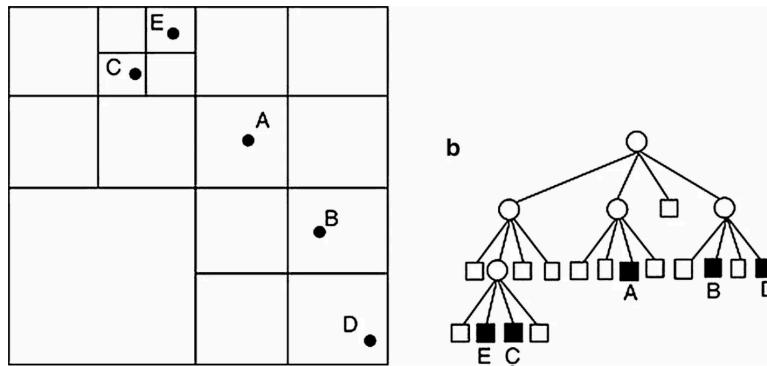
Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- a. Error blok berada di bawah threshold.
- b. Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah

dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar

1.3 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.

Tabel1. Metode Pengukuran Error

Metode	Formula
Variance	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	$\sigma_c^2 = \text{Variansi tiap kanal warna } c \text{ (R, G, B) dalam satu blok}$

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

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

2. Threshold (Ambang Batas)

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

3. Minimum Block Size (Ukuran Blok Minimum)

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

4. Compression Percentage (Persentase Kompresi) [BONUS]

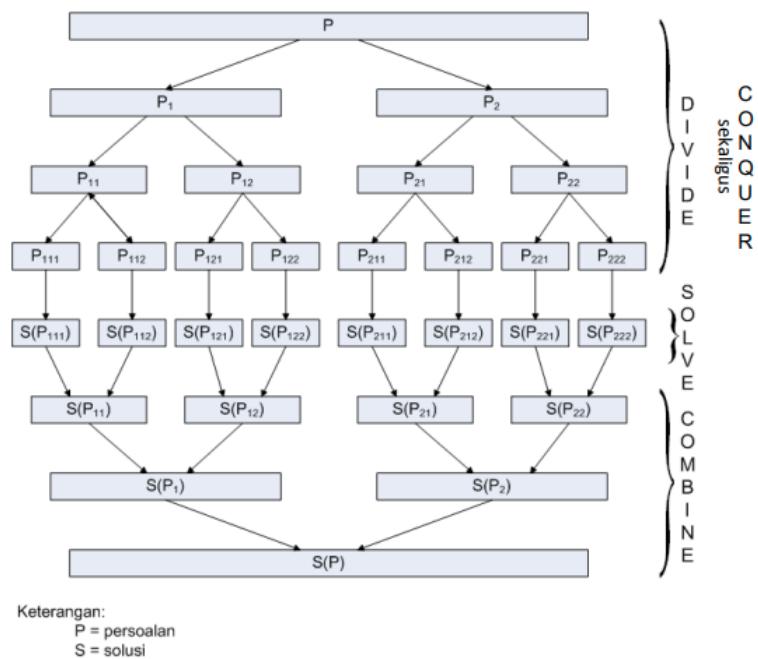
Presentasi kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

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

Bab II

Landasan Teori Algoritma Devide and Conquer

Algoritma devide and conquer adalah sebuah metode pemecahan masalah dengan 3 tahap. Tahapan yang pertama adalah tahap divide yaitu membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama). Kemudian tahapan tersebut dilanjutkan dengan tahapan conquer yaitu menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Tahapan terakhir dari algoritma ini adalah Combine. Setelah persoalan dibagi menjadi berukuran kecil untuk diselesaikan perbagian kecil, solusi keseluruhannya didapatkan dengan menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.



Gambar 4. Ilustrasi Algoritma Devide and Conquer

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

Algoritma divide and conquer memiliki skema umum seperti berikut:

```

procedure DIVIDEandCONQUER(input P : problem, n : integer)
{Menyelesaikan persoalan P dengan algoritma divide and conquer
Masukan: persoalan P berukuran n
Keluaran: solusi dari persoalan semula}

```

Deklarasi:

```

r : integer // jumlah sub-persoalan
Pi : array of problem // array sub-persoalan
ni : array of integer // array ukuran dari masing-masing sub-persoalan

```

Algoritma:

```

if n ≤ n0 then
    // Basis: ukuran persoalan cukup kecil untuk diselesaikan langsung
    SOLVE(P)
else
    // Divide: bagi persoalan menjadi r sub-persoalan
    DIVIDE P menjadi P1, P2, ..., Pr, masing-masing berukuran n1, n2, ..., nr

    for i ← 1 to r do
        DIVIDEandCONQUER(Pi[i], ni[i]) // Rekursif untuk tiap sub-persoalan
    endfor

    // Combine: gabungkan solusi dari sub-persoalan untuk membentuk solusi akhir
    COMBINE solusi dari P1, P2, ..., Pr menjadi solusi untuk P
endif

```

Algoritma Divide and Conquer memiliki kompleksitas waktu sebagai berikut:

$$T(n) = \begin{cases} g(n) & , n \leq n_0 \\ T(n_1) + T(n_2) \dots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

- $T(n)$: kompleksitas waktu penyelesaian persoalan P yang berukuran n

- $g(n)$: kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil
- $Tn1 + Tn2 \dots + Tnr$: kompleksitas waktu untuk memproses setiap upa-persoalan
- $f(n)$: kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan
- Tahap DIVIDE dapat dilakukan dalam $O(1)$, sehingga tidak dimasukkan ke dalam formula

Bab III

Metodologi Implementasi

3.1 Langkah-Langkah Devide and Conquer

3.1.1 Inisialisasi dan Persiapan Data

Tahap pertama dalam program adalah melakukan inisialisasi dan persiapan data yang diperlukan sebelum proses kompresi dimulai. Gambar input dibaca dari file menggunakan `BufferedImage`, dan dilakukan pengecekan ukuran gambar untuk mencegah beban memori yang terlalu besar. Jika gambar melebihi ambang batas (misalnya >10 megapixel), maka gambar akan diskalakan agar lebih kecil. Selain itu, jika pengguna memberikan target rasio kompresi, maka program secara otomatis akan mencari nilai ambang error (threshold) yang optimal untuk mencapai rasio tersebut. Proses ini dilakukan di kelas `ImageCompressor.java` sebelum membangun struktur Quadtree.

3.1.2 Divide – Membagi Gambar Menjadi Sub-Blok

Tahap divide merupakan inti dari metode **divide and conquer**, di mana gambar yang besar dibagi menjadi sub-blok yang lebih kecil. Proses ini dilakukan oleh fungsi `buildTree()` dalam kelas `Quadtree.java`. Pertama-tama, untuk setiap blok gambar, program menghitung rata-rata warna dan menghitung nilai error (ketidakhomogenan warna) dengan menggunakan metode yang dipilih pengguna seperti **Variance**, **MAD**, **Entropy**, atau **SSIM**. Jika nilai error suatu blok melebihi threshold dan ukuran blok masih lebih besar dari batas minimum (`minBlockSize`), maka blok tersebut dibagi menjadi empat bagian: top-left, top-right, bottom-left, dan bottom-right. Proses ini memastikan bahwa bagian gambar yang kompleks akan mendapatkan perhatian lebih detail dibanding bagian yang lebih homogen.

3.1.3 Conquer – Menyelesaikan Setiap Sub-Blok Secara Rekursif

Setelah gambar dibagi menjadi empat kuadran, masing-masing kuadran akan diproses secara rekursif. Artinya, setiap sub-blok akan dicek ulang apakah ia cukup homogen. Jika tidak, maka ia akan dibagi kembali menjadi empat bagian lebih kecil. Proses ini berlangsung secara berulang dan mendalam, mengikuti struktur pohon Quadtree hingga semua bagian gambar menjadi cukup homogen (atau ukuran minimumnya tercapai). Dengan cara ini, program secara otomatis menyesuaikan tingkat pembagian berdasarkan detail yang dimiliki oleh setiap bagian gambar. Bagian yang lebih kompleks akan menghasilkan lebih banyak node dan pembagian, sedangkan bagian yang sederhana akan berhenti lebih awal.

3.1.4 Combine – Menggabungkan Solusi Sub-Blok Menjadi Gambar Kompresi

Setelah semua sub-blok selesai diproses, langkah terakhir adalah menggabungkan semua hasil ke dalam gambar kompresi akhir. Proses ini dilakukan dengan mewarnai setiap node leaf dari pohon Quadtree dengan warna rata-rata bloknya. Fungsi `renderQuadtree()` akan menggambar blok-blok tersebut ke dalam `BufferedImage` baru yang mewakili versi terkompresi dari gambar asli. Hanya blok-blok leaf (yang tidak dibagi lagi) yang digunakan dalam gambar akhir. Dengan demikian, hasil akhir merupakan representasi visual dari pohon Quadtree, di mana setiap bagian gambar yang cukup homogen disederhanakan menjadi satu warna. Hal ini memungkinkan ukuran gambar menjadi lebih kecil namun tetap mempertahankan bentuk visual keseluruhan.

3.1.5 Penghitungan Statistik dan Penyimpanan Hasil

Setelah gambar berhasil dikompresi, program menghitung dan mencetak statistik hasil kompresi seperti ukuran file sebelum dan sesudah kompresi, persentase kompresi, kedalaman pohon Quadtree, jumlah node, dan waktu eksekusi. Statistik ini ditampilkan kepada pengguna dalam bentuk yang mudah dibaca melalui objek `CompressionStats`. Selain itu, program juga menyimpan hasil gambar kompresi ke file yang ditentukan oleh

pengguna, dan secara opsional bisa menyimpan GIF yang menunjukkan proses pembentukan pohon Quadtree dari awal hingga akhir.

3.1.6 Pseudocode

Kompresi Gambar Menggunakan Divide and Conquer (Quadtree)

Procedure CompressImage(image, threshold, minBlockSize, errorMethod)

```
root ← BuildTree(image, 0, 0, image.width, image.height, 0)
result ← CreateEmptyImage(image.width, image.height)
RenderQuadtree(result, root)
return result
EndProcedure
```

BuildTree (Divide dan Conquer)

Procedure BuildTree(image, x, y, width, height, depth)

```
avgColor ← CalculateAverageColor(image, x, y, width, height)
error ← CalculateError(image, x, y, width, height, errorMethod)

node ← NewNode(x, y, width, height, avgColor, error)

If error ≤ threshold OR width ≤ minBlockSize OR height ≤ minBlockSize Then
    node.isLeaf ← true
    return node
Else
    halfWidth ← width / 2
    halfHeight ← height / 2

    topLeft ← BuildTree(image, x, y, halfWidth, halfHeight, depth + 1)
    topRight ← BuildTree(image, x + halfWidth, y, halfWidth, halfHeight, depth + 1)
    bottomLeft ← BuildTree(image, x, y + halfHeight, halfWidth, halfHeight, depth +
    1)
    bottomRight ← BuildTree(image, x + halfWidth, y + halfHeight, halfWidth,
    halfHeight, depth + 1)
```

```

node.split(topLeft, topRight, bottomLeft, bottomRight)
node.isLeaf ← false
return node
EndIf
EndProcedure

```

RenderQuadtree (Combine)'

```

Procedure RenderQuadtree(image, node)
If node.isLeaf Then
    FillRectangle(image, node.x, node.y, node.width, node.height, node.avgColor)
Else
    RenderQuadtree(image, node.topLeft)
    RenderQuadtree(image, node.topRight)
    RenderQuadtree(image, node.bottomLeft)
    RenderQuadtree(image, node.bottomRight)
EndIf
EndProcedure

```

3.2 Source Program

3.2.1 Struktur File

```

bin/
doc/
└── Tucil2_13523153_18222130.pdf
src/
├── compression/
│   ├── CompressionStats.java
│   └── ImageCompressor.java
├── error/
│   ├── ErrorCalculator.java
│   └── ErrorMethod.java

```

```

model/
|   └── Node.java
|   └── Quadtree.java
└── util/
    ├── GifGenerator.java
    └── ImageUtil.java
└── Main.java

test/
└── compressed/
    ├── flowers_compressed.jpg
    └── starry_night_full.jpg
└── process/
    └── starry_night_full.gif
└── raw/
    ├── flowers.jpg
    ├── heart.jpg
    ├── starry_night_full.jpg
    └── wall-e.jpg
└── README.md

```

3.2.2 CompressionStats.java

Menyimpan dan menghitung statistik terkait proses kompresi, termasuk ukuran file asli dan terkompresi, kedalaman pohon, jumlah node, dan waktu eksekusi. Berisi metode untuk menghitung persentase kompresi dan menghasilkan ringkasan teks dari hasil.

```

package src.compression;

public class CompressionStats {
    private long originalFileSize;
    private long compressedFileSize;
    private int treeDepth;
}

```

```

private int nodeCount;
private long executionTimeMs;

// Store compression stats
public CompressionStats(long originalFileSize, long compressedFileSize, int
treeDepth, int nodeCount, long executionTimeMs){
    this.originalFileSize = originalFileSize;
    this.compressedFileSize = compressedFileSize;
    this.treeDepth = treeDepth;
    this.nodeCount = nodeCount;
    this.executionTimeMs = executionTimeMs;
}

// Calculate compression ratio
public double getCompressionPercentage(){
    if (originalFileSize == 0){
        return 0;
    }
    return 1.0 - ((double) compressedFileSize / originalFileSize);
}

// Stats getters
public int getTreeDepth(){ return treeDepth; }
public int getNodeCount(){ return nodeCount; }

// Format time display
private String formatTime(){
    if (executionTimeMs < 1000){
        return executionTimeMs + " ms";
    } else {
        double seconds = executionTimeMs / 1000.0;
        return String.format("%.2f seconds", seconds);
    }
}

```

```

    }

    //Format size display
    private static String formatSize(long size){
        if (size < 1024){
            return size + " bytes";
        } else if (size < 1024 * 1024){
            return String.format("%.2f KB", size / 1024.0);
        } else {
            return String.format("%.2f MB", size / (1024.0 * 1024));
        }
    }

    //Generate report text
    public String getSummary(){
        StringBuilder sb = new StringBuilder();
        sb.append("Compression Statistics:\n");
        sb.append("-----\n");
        sb.append("Execution time: ").append(formatTime()).append("\n");
                sb.append("Original      image      size:\n");
        sb.append(formatSize(originalFileSize)).append("\n");
                sb.append("Compressed     image      size:\n");
        sb.append(formatSize(compressedFileSize)).append("\n");
                sb.append("Compression percentage: ").append(String.format("%.2f%%",
        getCompressionPercentage() * 100)).append("\n");
        sb.append("Quadtree depth: ").append(treeDepth).append("\n");
        sb.append("Number of nodes: ").append(nodeCount).append("\n");

        return sb.toString();
    }
}

```

3.2.3 ImageCompressor.java

Kelas yang mengoordinasikan seluruh proses kompresi. Memuat gambar, menerapkan algoritma quadtree dengan parameter yang ditentukan, menyimpan gambar terkompresi, dan menghasilkan GIF animasi. Termasuk optimasi untuk gambar yang besar dan algoritma pencarian biner untuk menemukan threshold yang optimal untuk rasio kompresi target.

```
package src.compression;

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import src.error.ErrorMethod;
import src.model.Quadtree;
import src.util.GifGenerator;

public class ImageCompressor{
    private String inputPath;
    private String outputPath;
    private String gifPath;
    private ErrorMethod errorMethod;
    private double threshold;
    private int minBlockSize;
    private double targetCompressionRatio;
    private boolean generateGif;
    private Quadtree quadtree;

    // Constructor
    public ImageCompressor(
        String inputName,
        String outputName,
```

```

String gifName,
ErrorMethod errorMethod,
double threshold,
int minBlockSize,
double targetCompressionRatio){

    // Use paths as provided directly
    this.inputPath = inputPath;
    this.outputPath = outputPath;
    this.gifPath = gifPath;

    // Ensure directory exists for output path
    File outputFile = new File(outputPath);
    if (outputFile.getParentFile() != null){
        outputFile.getParentFile().mkdirs();
    }

    // Ensure directories exist for gif path
    if (gifPath != null && !gifPath.isEmpty()){
        File gifFile = new File(gifPath);
        if (gifFile.getParentFile() != null){
            gifFile.getParentFile().mkdirs();
        }
    }

    this.errorMethod = errorMethod;
    this.threshold = threshold;
    this.minBlockSize = minBlockSize;
    this.targetCompressionRatio = targetCompressionRatio;
    this.generateGif = gifPath != null && !gifPath.isEmpty();
}

// Main compression process

```

```

public CompressionStats compress() throws IOException{
    long startTime = System.currentTimeMillis();

    //Load image
    File inputFile = new File(inputPath);
    if (!inputFile.exists()){
        throw new IOException("Input file does not exist: " + inputFile);
    }

    BufferedImage original = ImageIO.read(inputFile);

    // Scale large images
    if (original.getWidth() * original.getHeight() > 10000000) { // > 10MP
        double scale = Math.sqrt(10000000.0 / (original.getWidth() *
original.getHeight()));
        int newWidth = (int)(original.getWidth() * scale);
        int newHeight = (int)(original.getHeight() * scale);
        System.out.println("Image is very large, scaling down for processing...");
        original = scaleImage(original, newWidth, newHeight);
    }

    //Auto-adjust threshold
    if (targetCompressionRatio > 0){
        threshold = findOptimalThreshold(original, targetCompressionRatio);
    }

    //Create quadtree
    this.quadtree = new Quadtree(original, minBlockSize, threshold, errorMethod,
generateGif);
    BufferedImage compressed = quadtree.compressImage();

    // Save output
    File outputFile = new File(outputPath);

```

```

String format = outputPath.substring(outputPath.lastIndexOf('.') + 1);
ImageIO.write(compressed, format, outputFile);

//Create GIF
if(generateGif && quadtree.getCompressionSteps() != null){
    GifGenerator.createGif(quadtree.getCompressionSteps(), gifPath);
}

// Return stats
long endTime = System.currentTimeMillis();
return new CompressionStats(
    inputFile.length(),
    outputFile.length(),
    quadtree.getDepth(),
    quadtree.getNodeCount(),
    endTime - startTime
);
}

//Find best threshold
private double findOptimalThreshold(BufferedImage original, double targetRatio){
    // Set search range
    double minThreshold = 0;
    double maxThreshold = 1000;
    double currentThreshold = (minThreshold + maxThreshold) / 2;
    double currentRatio;
    int maxIterations = 8;

    // Smaller test image
    BufferedImage testImage = scaleDown(original, 2);
    int testBlockSize = Math.max(1, minBlockSize / 2);
}

```

```

for (int i = 0; i < maxIterations; i++) {
    // Test compression
    Quadtree testTree = new Quadtree(testImage, testBlockSize,
        currentThreshold, errorMethod, false);

    // Check ratio
    currentRatio = 1.0 - (double) testTree.getNodeCount() /
        (testImage.getWidth() * testImage.getHeight());

    // Close enough
    if (Math.abs(currentRatio - targetRatio) < 0.05) {
        break;
    }

    // Adjust threshold
    else if (currentRatio < targetRatio) {
        minThreshold = currentThreshold;
    } else {
        maxThreshold = currentThreshold;
    }

    currentThreshold = (minThreshold + maxThreshold) / 2;

    // Free memory
    System.gc();
}

return currentThreshold;
}

// Scale by factor
private BufferedImage scaleDown(BufferedImage original, int factor) {
    int width = original.getWidth() / factor;
    int height = original.getHeight() / factor;
}

```

```

        return scaleImage(original, width, height);
    }

    // Resize image
    private BufferedImage scaleImage(BufferedImage original, int width, int height) {
        // Choose format
        int imageType = (width * height > 4000000) ?
            BufferedImage.TYPE_3BYTE_BGR :
            BufferedImage.TYPE_INT_RGB;

        BufferedImage scaled = new BufferedImage(width, height, imageType);

        Graphics2D g = scaled.createGraphics();
        g.drawImage(original, 0, 0, width, height, null);
        g.dispose();

        return scaled;
    }
}

```

3.2.4 ErrorCalculator.java

Berisi implementasi dari semua metode pengukuran error. Kelas ini mengkuantifikasi seberapa baik warna rata-rata mewakili blok piksel dalam gambar, menggunakan berbagai metrik matematika. Juga termasuk metode untuk menghitung warna rata-rata dari region gambar.

```

package src.error;

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

```

```

public class ErrorCalculator {

    // Select error method
    public static double calculateError(BufferedImage image, int x, int y, int width, int
height, ErrorMethod method) {
        int[] avgColor = calculateAvgColor(image, x, y, width, height);

        switch (method) {
            case VARIANCE:
                return calculateVariance(image, x, y, width, height, avgColor);
            case MAD:
                return calculateMAD(image, x, y, width, height, avgColor);
            case MAX_DIFF:
                return calculateMaxDiff(image, x, y, width, height);
            case ENTROPY:
                return calculateEntropy(image, x, y, width, height);
            case SSIM:
                return calculateSSIM(image, x, y, width, height, avgColor);
            default:
                return calculateVariance(image, x, y, width, height, avgColor);
        }
    }

    // Get block's average color
    public static int[] calculateAvgColor(BufferedImage image, int x, int y, int width,
int height) {
        long rSum = 0, gSum = 0, bSum = 0;
        int count = 0;

        for (int j = y; j < y + height; j++) {
            for (int i = x; i < x + width; i++) {
                Color pixel = new Color(image.getRGB(i, j));
                rSum += pixel.getRed();
            }
        }
    }
}

```

```

        gSum += pixel.getGreen();
        bSum += pixel.getBlue();
        count++;
    }
}

return new int[]{
    (int)(rSum / count),
    (int)(gSum / count),
    (int)(bSum / count)
};
}

// Calculate color variance
private static double calculateVariance(BufferedImage image, int x, int y, int width, int height, int[] avgColor){
    double rVariance = 0, gVariance = 0, bVariance = 0;
    int count = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color pixel = new Color(image.getRGB(i, j));
            rVariance += Math.pow(pixel.getRed() - avgColor[0], 2);
            gVariance += Math.pow(pixel.getGreen() - avgColor[1], 2);
            bVariance += Math.pow(pixel.getBlue() - avgColor[2], 2);
        }
    }

    rVariance /= count;
    gVariance /= count;
    bVariance /= count;

    return (rVariance + gVariance + bVariance) / 3;
}

```

```

}

// Mean absolute deviation
private static double calculateMAD(BufferedImage image, int x, int y, int width, int
height, int[] avgColor){
    double rMAD = 0, gMAD = 0, bMAD = 0;
    int count = width * height;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color pixel = new Color(image.getRGB(i, j));
            rMAD += Math.abs(pixel.getRed() - avgColor[0]);
            gMAD += Math.abs(pixel.getGreen() - avgColor[1]);
            bMAD += Math.abs(pixel.getBlue() - avgColor[2]);
        }
    }

    rMAD /= count;
    gMAD /= count;
    bMAD /= count;

    return (rMAD + gMAD + bMAD) / 3;
}

// Max color difference
private static double calculateMaxDiff(BufferedImage image, int x, int y, int width,
int height){
    int rMin = 255, gMin = 255, bMin = 255;
    int rMax = 0, gMax = 0, bMax = 0;

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color pixel = new Color(image.getRGB(i, j));

```

```

rMin = Math.min(rMin, pixel.getRed());
gMin = Math.min(gMin, pixel.getGreen());
bMin = Math.min(bMin, pixel.getBlue());

rMax = Math.max(rMax, pixel.getRed());
gMax = Math.max(gMax, pixel.getGreen());
bMax = Math.max(bMax, pixel.getBlue());
}

}

double rDiff = rMax - rMin;
double gDiff = gMax - gMin;
double bDiff = bMax - bMin;

return (rDiff + gDiff + bDiff) / 3;
}

// Color distribution entropy

private static double calculateEntropy(BufferedImage image, int x, int y, int width, int height){
    int[] rHistogram = new int[256];
    int[] gHistogram = new int[256];
    int[] bHistogram = new int[256];

    int totalPixels = width * height;

    // Reset histograms
    Arrays.fill(rHistogram, 0);
    Arrays.fill(gHistogram, 0);
    Arrays.fill(bHistogram, 0);

    // Count colors
}

```

```

for(int j = y; j < y + height; j++) {
    for(int i = x; i < x + width; i++) {
        Color pixel = new Color(image.getRGB(i, j));
        rHistogram[pixel.getRed()]++;
        gHistogram[pixel.getGreen()]++;
        bHistogram[pixel.getBlue()]++;
    }
}

// Calculate entropy
double rEntropy = 0, gEntropy = 0, bEntropy = 0;

for (int i = 0; i < 256; i++) {
    if (rHistogram[i] > 0) {
        double probability = (double) rHistogram[i] / totalPixels;
        rEntropy -= probability * (Math.log(probability) / Math.log(2));
    }

    if (gHistogram[i] > 0) {
        double probability = (double) gHistogram[i] / totalPixels;
        gEntropy -= probability * (Math.log(probability) / Math.log(2));
    }

    if (bHistogram[i] > 0) {
        double probability = (double) bHistogram[i] / totalPixels;
        bEntropy -= probability * (Math.log(probability) / Math.log(2));
    }
}

return (rEntropy + gEntropy + bEntropy) / 3;
}

// Structural similarity index

```

```

private static double calculateSSIM(BufferedImage image, int x, int y, int width,
int height, int[] avgColor) {
    final double C1 = Math.pow(0.01 * 255, 2);
    final double C2 = Math.pow(0.03 * 255, 2);

    BufferedImage avgImage = new BufferedImage(width, height,
    BufferedImage.TYPE_INT_RGB);
    int avgRGB = new Color(avgColor[0], avgColor[1], avgColor[2]).getRGB();
    for (int j = 0; j < height; j++) {
        for (int i = 0; i < width; i++) {
            avgImage.setRGB(i, j, avgRGB);
        }
    }

    double[] meanX = new double[3];
    double[] meanY = new double[3];
    double[] varX = new double[3];
    double[] covarXY = new double[3];

    meanX[0] = meanY[0] = avgColor[0];
    meanX[1] = meanY[1] = avgColor[1];
    meanX[2] = meanY[2] = avgColor[2];

    for (int j = y; j < y + height; j++) {
        for (int i = x; i < x + width; i++) {
            Color pixel = new Color(image.getRGB(i, j));

            varX[0] += Math.pow(pixel.getRed() - meanX[0], 2);
            varX[1] += Math.pow(pixel.getGreen() - meanX[1], 2);
            varX[2] += Math.pow(pixel.getBlue() - meanX[2], 2);

            covarXY[0] += (pixel.getRed() - meanX[0]) * (avgColor[0] - meanY[0]);
            covarXY[1] += (pixel.getGreen() - meanX[1]) * (avgColor[1] - meanY[1]);
        }
    }
}

```

```

        covarXY[2] += (pixel.getBlue() - meanX[2]) * (avgColor[2] - meanY[2]);
    }
}

int n = width * height;
for (int i = 0; i < 3; i++) {
    varX[i] /= n;
    covarXY[i] /= n;
}

double[] varY = {0, 0, 0};

double[] ssim = new double[3];
for (int i = 0; i < 3; i++) {
    ssim[i] = ((2 * meanX[i] * meanY[i] + C1) * (2 * covarXY[i] + C2)) /
        ((meanX[i] * meanX[i] + meanY[i] * meanY[i] + C1) * (varX[i] + varY[i] + C2));
}

double ssimRGB = (ssim[0] + ssim[1] + ssim[2]) / 3;

return 1 - ssimRGB;
}
}

```

3.2.5 ErrorMethod.java

Enum yang mendefinisikan berbagai metode pengukuran error yang didukung, seperti Variance, MAD (Mean Absolute Difference), MAX_DIFF (Maximum Color Difference), Entropy, dan SSIM (Structural Similarity Index). Termasuk metode pembantu untuk konversi antara ID numerik dan tipe enum.

```
package src.error;
```

```
// Error method options
public enum ErrorMethod {
    VARIANCE(1, "Variance"),
    MAD(2, "Mean Absolute Deviation"),
    MAX_DIFF(3, "Max Pixel Difference"),
    ENTROPY(4, "Entropy"),
    SSIM(5, "Structural Similarity Index (Bonus)");

    private final int id;
    private final String name;

    // Constructor
    ErrorMethod(int id, String name) {
        this.id = id;
        this.name = name;
    }

    // Get method id
    public int getId() {
        return id;
    }

    // Get method name
    public String getName() {
        return name;
    }

    // Find by id
    public static ErrorMethod getById(int id) {
        for (ErrorMethod method : values()) {
            if (method.getId() == id) {
                return method;
            }
        }
    }
}
```

```

    }

    return null;
}

// List all methods
public static String getAvailableMethods(){
    StringBuilder sb = new StringBuilder();
    sb.append("Available error measurement methods:\n");
    for(ErrorMethod method : values()){
        sb.append(method.getId()).append(".");
    }.append(method.getName()).append("\n");
}
return sb.toString();
}
}

```

3.2.6 Node.java

Merepresentasikan node dalam struktur quadtree. Setiap node menyimpan posisinya (x, y, lebar, tinggi), warna rata-rata dari regionnya, nilai error, dan referensi ke empat anaknya (kiri-atas, kanan-atas, kiri-bawah, kanan-bawah). Node dapat berupa daun (mewakili satu blok warna) atau node internal dengan empat anak.

```

package src.model;

import java.awt.Color;

public class Node {
    // Region info
    private int x, y, width, height;
    private int[] avgColor;
    private double error;
}

```

```
// Children
private Node topLeft;
private Node topRight;
private Node bottomLeft;
private Node bottomRight;

// Leaf status
private boolean isLeaf;

public Node(int x, int y, int width, int height, int[] avgColor, double error){
    this.x = x;
    this.y = y;
    this.width = width;
    this.height = height;
    this.avgColor = avgColor;
    this.error = error;
    this.isLeaf = true;
}

// Position getters
public int getX(){ return x; }
public int getY(){ return y; }
public int getWidth(){ return width; }
public int getHeight(){ return height; }
public int[] getAvgColor(){ return avgColor; }
public double getError(){ return error; }
public boolean isLeaf(){ return isLeaf; }

// Get color as Color
public Color getColor(){
    return new Color(avgColor[0], avgColor[1], avgColor[2]);
}
```

```

// Child node getters

public Node getTopLeft(){ return topLeft; }

public Node getTopRight(){ return topRight; }

public Node getBottomLeft(){ return bottomLeft; }

public Node getBottomRight(){ return bottomRight; }

// Node state modifiers

public void setLeaf(boolean isLeaf){ this.isLeaf = isLeaf; }

public void setTopLeft(Node node){ this.topLeft = node; }

public void setTopRight(Node node){ this.topRight = node; }

public void setBottomLeft(Node node){ this.bottomLeft = node; }

public void setBottomRight(Node node){ this.bottomRight = node; }

// Split into four children

public void split(Node topLeft, Node topRight, Node bottomLeft, Node
bottomRight){

    this.topLeft = topLeft;

    this.topRight = topRight;

    this.bottomLeft = bottomLeft;

    this.bottomRight = bottomRight;

    this.isLeaf = false;

}

}

```

3.2.7 Quadtree.java

Mengimplementasikan struktur data quadtree inti menggunakan algoritma divide-and-conquer. File ini membangun pohon secara rekursif dengan membagi gambar menjadi empat kuadran, menghitung error untuk setiap region, dan terus membagi jika error melebihi threshold. Kelas ini juga menangani rendering pohon ke gambar terkompresi dan mengambil frame untuk animasi GIF.

```
package src.model;
```

```
import java.awt.AlphaComposite;
import java.awt.BasicStroke;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;
import src.error.ErrorCalculator;
import src.error.ErrorMethod;

public class Quadtree{
    private Node root;
    private int minBlockSize;
    private double threshold;
    private ErrorMethod errorMethod;
    private int depth;
    private int nodeCount;
    private BufferedImage originalImage;
    private List<BufferedImage> compressionSteps;
    private int stepCounter = 0;
    private static final int MAX_FRAMES = 25;

    public Quadtree(BufferedImage image, int minBlockSize, double threshold,
                    ErrorMethod errorMethod, boolean captureSteps) {
        this.minBlockSize = minBlockSize;
        this.threshold = threshold;
        this.errorMethod = errorMethod;
        this.depth = 0;
        this.nodeCount = 0;
        this.originalImage = image;
        this.compressionSteps = captureSteps ? new ArrayList<>() : null;
    }
}
```

```

// Add first frame
if(captureSteps){
    BufferedImage initialImage = new BufferedImage(
        image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
    Graphics2D g2d = initialImage.createGraphics();
    g2d.drawImage(image, 0, 0, null);
    g2d.dispose();

    compressionSteps.add(initialImage);
}

// Build tree
this.root = buildTree(image, 0, 0, image.getWidth(), image.getHeight(), 0);

// Add last frame
if(captureSteps){
    BufferedImage finalImage = new BufferedImage(
        image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
    renderQuadtree(finalImage, root);
    compressionSteps.add(finalImage);
}
}

private Node buildTree(BufferedImage image, int x, int y, int width, int height, int
currentDepth){
    //Track stats
    this.nodeCount++;
    this.depth = Math.max(this.depth, currentDepth);

    //Get block info
    int[] avgColor = ErrorCalculator.calculateAvgColor(image, x, y, width, height);
    double error = ErrorCalculator.calculateError(image, x, y, width, height,
errorMethod);
}

```

```

// Create node
Node node = new Node(x, y, width, height, avgColor, error);

// Split if needed
if (error > threshold && width > minBlockSize && height > minBlockSize) {
    int halfWidth = width / 2;
    int halfHeight = height / 2;

    // Create children
    Node topLeft = buildTree(image, x, y, halfWidth, halfHeight, currentDepth + 1);
    Node topRight = buildTree(image, x + halfWidth, y, halfWidth, halfHeight,
        currentDepth + 1);
    Node bottomLeft = buildTree(image, x, y + halfHeight, halfWidth, halfHeight,
        currentDepth + 1);
    Node bottomRight = buildTree(image, x + halfWidth, y + halfHeight, halfWidth,
        halfHeight, currentDepth + 1);

    // Connect children
    node.split(topLeft, topRight, bottomLeft, bottomRight);

    // Capture frames
    int captureFrequency = calculateCaptureFrequency(currentDepth);
    if (compressionSteps != null && stepCounter % captureFrequency == 0 &&
        compressionSteps.size() < MAX_FRAMES) {
        captureProgressFrame();
    }
    stepCounter++;
}

return node;
}

```

```

// Frame capture frequency
private int calculateCaptureFrequency(int depth){
    // Early splits
    if (depth <= 2) return 1;

    // Based on size
    int totalPixels = originalImage.getWidth() * originalImage.getHeight();
    if (totalPixels < 250000){
        return 2;
    } else if (totalPixels < 1000000){
        return 4;
    } else {
        return 8;
    }
}

// Capture compression progress
private void captureProgressFrame(){
    try{
        // Create new image
        BufferedImage stepImage = new BufferedImage(
            originalImage.getWidth(), originalImage.getHeight(),
            BufferedImage.TYPE_INT_RGB);

        // Draw background
        Graphics2D g = stepImage.createGraphics();

        g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
        0.3f));
        g.drawImage(originalImage, 0, 0, null);

        // Draw quadtree
    }
}

```

```

g.setComposite(AlphaComposite.getInstance(AlphaComposite.SRC_OVER,
1.0f));
    renderQuadtreeWithStrongBorders(stepImage, root, g);
    g.dispose();

    // Save frame
    compressionSteps.add(stepImage);

    // Free memory
    if (originalImage.getWidth() * originalImage.getHeight() > 1000000){
        System.gc();
    }
} catch(OutOfMemoryError e){
    System.out.println("Warning: Memory limit reached, stopping frame
capture");
    compressionSteps = null;
}
}

// Draw with thick borders
private void renderQuadtreeWithStrongBorders(BufferedImage image, Node
node, Graphics2D g){
    if (node == null) return;

    if (node.isLeaf()){
        // Fill block
        g.setColor(node.getColor());
        g.fillRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());

        // Draw border
        g.setColor(new Color(0, 0, 0, 200));
        g.setStroke(new BasicStroke(Math.max(1, node.getWidth()/100)));
        g.drawRect(node.getX(), node.getY(), node.getWidth()-1,

```

```

node.getHeight()-1);
} else {
    // Border for parent
    if (node.getWidth() > minBlockSize * 4){
        g.setColor(new Color(255, 0, 0, 100));
        g.setStroke(new BasicStroke(Math.max(2, node.getWidth()/80)));
        g.drawRect(node.getX(), node.getY(), node.getWidth()-1,
node.getHeight()-1);
    }

    // Draw children
    if (node.getTopLeft() != null) renderQuadtreeWithStrongBorders(image,
node.getTopLeft(), g);
    if (node.getTopRight() != null) renderQuadtreeWithStrongBorders(image,
node.getTopRight(), g);
    if (node.getBottomLeft() != null) renderQuadtreeWithStrongBorders(image,
node.getBottomLeft(), g);
    if (node.getBottomRight() != null) renderQuadtreeWithStrongBorders(image,
node.getBottomRight(), g);
}

}

// Draw with thin borders
private void renderQuadtreeWithBorders(BufferedImage image, Node node,
Graphics2D g){
    if (node == null) return;

    if (node.isLeaf()){
        // Fill block
        g.setColor(node.getColor());
        g.fillRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());

        // Draw border
    }
}

```

```

        g.setColor(new Color(0, 0, 0, 128));
        g.drawRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());
    } else{
        // Draw children
        if (node.getTopLeft() != null) renderQuadtreeWithBorders(image,
node.getTopLeft(), g);
        if (node.getTopRight() != null) renderQuadtreeWithBorders(image,
node.getTopRight(), g);
        if (node.getBottomLeft() != null) renderQuadtreeWithBorders(image,
node.getBottomLeft(), g);
        if (node.getBottomRight() != null) renderQuadtreeWithBorders(image,
node.getBottomRight(), g);
    }
}

// Create compressed image
public BufferedImage compressImage(){
    int width = originalImage.getWidth();
    int height = originalImage.getHeight();

    // Choose format
    int imageType = (width * height > 4000000) ?
        BufferedImage.TYPE_3BYTE_BGR:
        BufferedImage.TYPE_INT_RGB;

    BufferedImage result = new BufferedImage(width, height, imageType);

    // Process large images in tiles
    if (width * height > 8000000){
        int tileSize = 512;
        for (int y = 0; y < height; y += tileSize){
            for (int x = 0; x < width; x += tileSize){
                int tileWidth = Math.min(tileSize, width - x);

```

```

        int tileHeight = Math.min(tileSize, height - y);
        renderTile(result, root, x, y, tileSize, tileHeight);

        // Free memory
        if (x % 1024 == 0 && y % 1024 == 0){
            System.gc();
        }
    }
}

} else{
    // Render at once
    renderQuadtree(result, root);
}

return result;
}

// Render part of image
private void renderTile(BufferedImage image, Node node, int tileX, int tileY, int
tileWidth, int tileHeight){
    if (node == null) return;

    // Skip non-intersecting
    if (!intersects(node.getX(), node.getY(), node.getWidth(), node.getHeight(),
tileX, tileY, tileWidth, tileHeight)){
        return;
    }

    if (node.isLeaf()){
        // Draw leaf intersection
        int x1 = Math.max(node.getX(), tileX);
        int y1 = Math.max(node.getY(), tileY);
        int x2 = Math.min(node.getX() + node.getWidth(), tileX + tileWidth);
    }
}

```

```

int y2 = Math.min(node.getY() + node.getHeight(), tileY + tileHeight);

if (x2 > x1 && y2 > y1) {
    Graphics2D g = image.createGraphics();
    g.setColor(node.getColor());
    g.fillRect(x1, y1, x2 - x1, y2 - y1);
    g.dispose();
}
} else {
    // Check children
    renderTile(image, node.getTopLeft(), tileX, tileY, tileSize, tileHeight);
    renderTile(image, node.getTopRight(), tileX, tileY, tileSize, tileHeight);
    renderTile(image, node.getBottomLeft(), tileX, tileY, tileSize, tileHeight);
    renderTile(image, node.getBottomRight(), tileX, tileY, tileSize, tileHeight);
}
}

// Check rectangle overlap
private boolean intersects(int x1, int y1, int w1, int h1, int x2, int y2, int w2, int h2){
    return x1 < x2 + w2 && x1 + w1 > x2 && y1 < y2 + h2 && y1 + h1 > y2;
}

// Render full quadtree
private void renderQuadtree(BufferedImage image, Node node){
    if (node == null) return;

    if (node.isLeaf()) {
        // Fill block
        Graphics2D g = image.createGraphics();
        g.setColor(node.getColor());
        g.fillRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());
        g.dispose();
    } else{

```

```

// Draw children
renderQuadtree(image, node.getTopLeft());
renderQuadtree(image, node.getTopRight());
renderQuadtree(image, node.getBottomLeft());
renderQuadtree(image, node.getBottomRight());
}

}

// Render with null check
private void renderQuadtreePartial(BufferedImage image, Node node){
    if (node == null) return;

    if (node.isLeaf()){
        // Fill block
        Graphics2D g = image.createGraphics();
        g.setColor(node.getColor());
        g.fillRect(node.getX(), node.getY(), node.getWidth(), node.getHeight());
        g.dispose();
    } else{
        // Check then draw
        if (node.getTopLeft() != null) renderQuadtreePartial(image,
node.getTopLeft());
        if (node.getTopRight() != null) renderQuadtreePartial(image,
node.getTopRight());
        if (node.getBottomLeft() != null) renderQuadtreePartial(image,
node.getBottomLeft());
        if (node.getBottomRight() != null) renderQuadtreePartial(image,
node.getBottomRight());
    }
}

// Get data methods
public List<BufferedImage> getCompressionSteps() { return

```

```
compressionSteps; }

    public int getDepth(){ return depth; }

    public int getNodeCount(){ return nodeCount; }

    public Node getRoot(){ return root; }

}
```

3.2.8 GifGenerator.java

Membuat animasi GIF dari serangkaian frame yang menunjukkan proses pembentukan quadtree dan kompresi gambar. Mencakup pengoptimalan untuk mengelola memori dengan gambar besar, menskalakan frame, membatasi jumlah frame, dan menambahkan overlay informasi yang menunjukkan kemajuan kompresi.

```
package src.util;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageTypeSpecifier;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.stream.ImageOutputStream;

public class GifGenerator {
```

```

public static void createGif(List<BufferedImage> frames, String outputPath)
throws IOException {
    if (frames == null || frames.isEmpty()) {
        throw new IllegalArgumentException("No frames provided");
    }

    System.out.println("Creating GIF with " + frames.size() + " frames...");

    // Get image size
    BufferedImage firstFrame = frames.get(0);
    int width = firstFrame.getWidth();
    int height = firstFrame.getHeight();

    // Scale large images
    double scale = 1.0;
    if (width * height > 1000000) { // > 1MP
        scale = Math.sqrt(1000000.0 / (width * height));
        System.out.println("Scaling GIF to " + (int)(scale * 100) + "% to fit memory
constraints");
    }

    // Process frames
    List<BufferedImage> processedFrames = preprocessFrames(frames, scale);
    System.out.println("Processed " + processedFrames.size() + " frames for GIF");

    // Get GIF writer
    ImageWriter writer = ImageIO.getImageWritersByFormatName("gif").next();

    // Setup output
    File outputFile = new File(outputPath);
    ImageOutputStream ios = ImageIO.createImageOutputStream(outputFile);
    writer.setOutput(ios);
}

```

```

// Set GIF params
ImageWriteParam params = writer.getDefaultWriteParam();
    ImageTypeSpecifier typeSpec = 
ImageTypeSpecifier.createFromBufferedImageType(BufferedImage.TYPE_INT_RGB);

// Setup animation
IIOMetadata metadata = writer.getDefaultImageMetadata(typeSpec, params);
String metaFormat = metadata.getNativeMetadataFormatName();
    IIOMetadataNode root = (IIOMetadataNode)
metadata.getAsTree(metaFormat);

// Set frame timing
IIOMetadataNode gce = getNode(root, "GraphicControlExtension");
gce.setAttribute("disposalMethod", "none");
gce.setAttribute("userInputFlag", "FALSE");
gce.setAttribute("transparentColorFlag", "FALSE");
gce.setAttribute("delayTime", "30"); // 0.3 sec
gce.setAttribute("transparentColorIndex", "0");

// Enable looping
IIOMetadataNode appExtensions = getNode(root, "ApplicationExtensions");
IIOMetadataNode appExt = new IIOMetadataNode("ApplicationExtension");
appExt.setAttribute("applicationID", "NETSCAPE");
appExt.setAttribute("authenticationCode", "2.0");
appExt.setUserObject(new byte[] {1, 0, 0});
appExtensions.appendChild(appExt);

metadata.setFromTree(metaFormat, root);

// Final frame timing
IIOMetadata finalFrameMetadata =
writer.getDefaultImageMetadata(typeSpec, params);

```

```

        IIOMetadataNode finalRoot = (IIOMetadataNode)
finalFrameMetadata.getAsTree(metaFormat);

        IIOMetadataNode finalGce = getNode(finalRoot, "GraphicControlExtension");
        finalGce.setAttribute("disposalMethod", "none");
        finalGce.setAttribute("userInputFlag", "FALSE");
        finalGce.setAttribute("transparentColorFlag", "FALSE");
        finalGce.setAttribute("delayTime", "300"); // 3 sec
        finalGce.setAttribute("transparentColorIndex", "0");

        // Final frame looping
        IIOMetadataNode finalAppExtensions = getNode(finalRoot,
"ApplicationExtensions");
        IIOMetadataNode finalAppExt = new
IIOMetadataNode("ApplicationExtension");
        finalAppExt.setAttribute("applicationID", "NETSCAPE");
        finalAppExt.setAttribute("authenticationCode", "2.0");
        finalAppExt.setUserObject(new byte[] {1, 0, 0});
        finalAppExtensions.appendChild(finalAppExt);

        finalFrameMetadata.setFromTree(metaFormat, finalRoot);

        // Start sequence
writer.prepareWriteSequence(null);

        System.out.println("Writing frames to GIF...");
        // Write normal frames
        for (int i = 0; i < processedFrames.size() - 1; i++) {
            writer.writeToSequence(new IIOImage(processedFrames.get(i), null,
metadata), params);

        // Clear memory
        if (i % 5 == 0 && width * height > 500000) {
            processedFrames.set(i, null);

```

```

        System.gc();
    }

}

//Write final frame
if (processedFrames.size() > 0){
    writer.writeToSequence(
        new IIOImage(processedFrames.get(processedFrames.size() - 1), null,
finalFrameMetadata),
        params
    );
}

//Clean up
writer.endWriteSequence();
ios.close();
writer.dispose();

System.out.println("GIF created successfully at: " + outputPath);
}

private static List<BufferedImage> preprocessFrames(List<BufferedImage>
originalFrames, double scale){
    if (originalFrames.size() <= 2){
        return originalFrames; //Too few
    }

//Get key frames
    BufferedImage firstFrame = originalFrames.get(0);
    BufferedImage lastFrame = originalFrames.get(originalFrames.size() - 1);

//Create result list
    List<BufferedImage> result = new ArrayList<>();
}

```

```

// Limit frame count
int maxFrames = Math.min(25, originalFrames.size());
int step = originalFrames.size() / maxFrames;
if (step < 1) step = 1;

// Add first frame
result.add(scaleImage(firstFrame, scale));

// Add middle frames
for (int i = step; i < originalFrames.size() - 1; i += step){
    result.add(scaleImage(originalFrames.get(i), scale));
}

// Add final frame
result.add(scaleImage(lastFrame, scale));

// Add frame info
return addFrameInfo(result);
}

private static BufferedImage scaleImage(BufferedImage source, double scale){
    // Skip if no scaling
    if (scale >= 0.99 || scale <= 0){
        return source;
    }

    int newWidth = (int)(source.getWidth() * scale);
    int newHeight = (int)(source.getHeight() * scale);

    // Choose image type
    int imageType = (newWidth * newHeight > 1000000) ?
        BufferedImage.TYPE_3BYTE_BGR:

```

```

        BufferedImage.TYPE_INT_RGB;

        BufferedImage scaled = new BufferedImage(newWidth, newHeight,
imageType);

        Graphics2D g = scaled.createGraphics();
        g.drawImage(source, 0, 0, newWidth, newHeight, null);
        g.dispose();

        return scaled;
    }

    private static List<BufferedImage> addFrameInfo(List<BufferedImage> frames)
{
    // Add overlays
    for (int i = 0; i < frames.size(); i++) {
        BufferedImage frame = frames.get(i);
        Graphics2D g = frame.createGraphics();

        // Info box
        int boxWidth = 200;
        int boxHeight = 50;
        int boxX = 10;
        int boxY = 10;

        // Draw background
        g.setColor(new Color(0, 0, 0, 150));
        g.fillRect(boxX, boxY, boxWidth, boxHeight);

        // Draw frame info
        g.setColor(Color.WHITE);
        String frameInfo = "Frame " + (i+1) + " of " + frames.size();
        g.drawString(frameInfo, boxX + 10, boxY + 20);
    }
}

```

```

// Show progress
String progressLabel = i == 0 ? "Original Image" :
    i == frames.size()-1 ? "Final Compression" :
        "Quadtree Formation " + Math.round((i/(float)frames.size()-1)*100) +
"%",
g.drawString(progressLabel, boxX + 10, boxY + 40);

// Progress bar
int barWidth = boxWidth - 20;
int barHeight = 8;
int barX = boxX + 10;
int barY = boxY + boxHeight - barHeight - 5;

// Bar background
g.setColor(Color.DARK_GRAY);
g.fillRect(barX, barY, barWidth, barHeight);

// Progress fill
g.setColor(Color.GREEN);
int fillWidth = (int)(barWidth * (i / (float)(frames.size() - 1)));
g.fillRect(barX, barY, fillWidth, barHeight);

g.dispose();
}

return frames;
}

private static IIOMetadataNode getNode(IIOMetadataNode root, String name){
    for (int i = 0; i < root.getLength(); i++){
        if (root.item(i).getNodeName().equalsIgnoreCase(name)){
            return (IIOMetadataNode) root.item(i);
        }
    }
}

```

```

    }

}

IIOMetadataNode node = new IIOMetadataNode(name);
root.appendChild(node);
return node;
}
}

```

3.2.9 ImageUtil.java

Menyediakan fungsi utilitas untuk operasi gambar umum, seperti memvalidasi file gambar, mendapatkan dimensi gambar, menyalin gambar, dan menggambar grid untuk visualisasi quadtree.

```

package src.util;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class ImageUtil {

    // Check file is image
    public static boolean isValidImageFile(String path) {
        try {
            File file = new File(path);
            if (!file.exists()) {
                return false;
            }
        }
    }
}

```

```

        return ImageIO.read(file) != null;
    } catch (IOException e) {
        return false;
    }
}

// Get width and height
public static int[] getImageDimensions(String path) throws IOException{
    BufferedImage image = ImageIO.read(new File(path));
    return new int[] {image.getWidth(), image.getHeight()};
}

// Clone image
public static BufferedImage copyImage(BufferedImage source){
    BufferedImage copy = new BufferedImage(
        source.getWidth(), source.getHeight(), source.getType());
    Graphics2D g = copy.createGraphics();
    g.drawImage(source, 0, 0, null);
    g.dispose();
    return copy;
}

// Draw grid lines
public static void drawQuadtreeGrid(BufferedImage image, int x, int y, int width,
int height, Color color){
    Graphics2D g = image.createGraphics();
    g.setColor(color);
    g.drawRect(x, y, width, height);
    g.dispose();
}
}

```

3.2.10 Main.java

Titik masuk utama aplikasi yang menangani interaksi pengguna. File ini mengumpulkan parameter input (jalur gambar input, metode pengukuran error, nilai threshold, ukuran blok minimal, rasio kompresi target, dan jalur output), meneruskannya ke ImageCompressor, dan menampilkan hasil kompresi. Termasuk validasi input yang kuat dan saran yang bermanfaat untuk nilai parameter.

```
import java.io.File;
import java.util.Scanner;
import src.compression.CompressionStats;
import src.compression.ImageCompressor;
import src.error.ErrorMethod;
import src.util.ImageUtil;

public class Main {

    public static void main(String[] args) {
        System.out.println();
        System.out.println("== Quadtree Image Compression ==");
        System.out.println("IF2211 - Strategi Algoritma - 13523153 & 18222130");
        System.out.println("-----");

        Scanner scanner = new Scanner(System.in);

        try {
            // Get user inputs
            String inputPath = getInputPath(scanner);
            ErrorMethod errorMethod = getErrorMethod(scanner);
            double threshold = getThreshold(scanner, errorMethod);
            int minBlockSize = getMinBlockSize(scanner);
            double targetRatio = getTargetCompressionRatio(scanner);
            String outputPath = getOutputPath(scanner);
        }
    }
}
```

```

String gifPath = getGifPath(scanner);

// Compress image
System.out.println("\nStarting image compression...");
ImageCompressor compressor = new ImageCompressor(
    inputPath, outputPath, gifPath, errorMethod, threshold, minBlockSize,
targetRatio
);

CompressionStats stats = compressor.compress();

// Show results
System.out.println("\n" + stats.getSummary());
System.out.println("Compressed image saved to: " + outputPath);

if (gifPath != null && !gifPath.isEmpty()){
    System.out.println("Compression process GIF saved to: " + gifPath);
}

} catch (Exception e){
    System.err.println("Error: " + e.getMessage());
    e.printStackTrace();
} finally{
    scanner.close();
}
}

// Get image path
private static String getInputPath(Scanner scanner){
    String path = "";
    boolean valid = false;

    while (!valid){

```

```

System.out.print("Enter the absolute path of the image to compress: ");
path = scanner.nextLine().trim();

if (path.isEmpty()){
    System.out.println("Error: Path cannot be empty.");
    continue;
}

File file = new File(path);
if (!file.exists()){
    System.out.println("Error: File does not exist.");
    continue;
}

if (!ImageUtil.isValidImageFile(path)){
    System.out.println("Error: Not a valid image file.");
    continue;
}

valid = true;
}

return path;
}

// Select error method
private static ErrorMethod getErrorMethod(Scanner scanner){
    ErrorMethod method = null;

    while (method == null){
        System.out.println("\n" + ErrorMethod.getAvailableMethods());
        System.out.print("Select error measurement method (1-"
        ErrorMethod.values().length + "): ");
}

```

```

try{
    int choice = Integer.parseInt(scanner.nextLine().trim());
    method = ErrorMethod.getById(choice);

    if(method == null){
        System.out.println("Error: Invalid choice. Please select a number from the
list.");
    }
} catch (NumberFormatException e){
    System.out.println("Error: Please enter a valid number.");
}
}

return method;
}

// Set error threshold
private static double getThreshold(Scanner scanner, ErrorMethod errorMethod){
    double threshold = 0;
    boolean valid = false;

    // Range suggestions
    String hint;
    switch(errorMethod){
        case VARIANCE: hint = "Suggested range: 10-1000"; break;
        case MAD:     hint = "Suggested range: 5-100"; break;
        case MAX_DIFF: hint = "Suggested range: 10-200"; break;
        case ENTROPY:   hint = "Suggested range: 0.1-5.0"; break;
        case SSIM:     hint = "Suggested range: 0.01-0.5"; break;
        default:      hint = "Enter a positive number";
    }
}

```

```

while (!valid){
    System.out.println("\nEnter the threshold value (" + hint + "):");
    System.out.print("Threshold: ");

    try{
        threshold = Double.parseDouble(scanner.nextLine().trim());

        if (threshold <= 0){
            System.out.println("Error: Threshold must be positive.");
            continue;
        }

        valid = true;
    } catch (NumberFormatException e){
        System.out.println("Error: Please enter a valid number.");
    }
}

return threshold;
}

// Set min block size
private static int getMinBlockSize(Scanner scanner){
    int size = 0;
    boolean valid = false;

    while (!valid){
        System.out.print("\nEnter the minimum block size (2, 4, 8, 16, etc.): ");

        try{
            size = Integer.parseInt(scanner.nextLine().trim());
            if (size < 1){
                System.out.println("Error: Block size must be greater than 0.");
            } else {
                valid = true;
            }
        } catch (NumberFormatException e){
            System.out.println("Error: Please enter a valid number.");
        }
    }

    return size;
}

```

```

        System.out.println("Error: Minimum block size must be at least 1.");
        continue;
    }

    valid = true;
} catch (NumberFormatException e){
    System.out.println("Error: Please enter a valid number.");
}
}

return size;
}

// Set compression ratio
private static double getTargetCompressionRatio(Scanner scanner){
    double ratio = 0;
    boolean valid = false;

    while (!valid){
        System.out.println("\nEnter target compression ratio (0–1.0, where 1.0 =\n100% compression):");
        System.out.println("Enter 0 to disable automatic threshold adjustment.");
        System.out.print("Target ratio: ");

        try{
            ratio = Double.parseDouble(scanner.nextLine().trim());

            if (ratio < 0 || ratio > 1){
                System.out.println("Error: Ratio must be between 0 and 1.0.");
                continue;
            }
        }

        valid = true;
    }
}

```

```

        } catch (NumberFormatException e){
            System.out.println("Error: Please enter a valid number.");
        }
    }

    return ratio;
}

// Set output path
private static String getOutputPath(Scanner scanner){
    String path = "";
    boolean valid = false;

    while (!valid){
        System.out.print("\nEnter the absolute path for the compressed image
output: ");
        path = scanner.nextLine().trim();

        if (path.isEmpty()){
            System.out.println("Error: Path cannot be empty.");
            continue;
        }

        File file = new File(path);
        File parentDir = file.getParentFile();

        if (parentDir != null && !parentDir.exists()){
            System.out.println("Warning: Directory does not exist. Create? (y/n)");
            String response = scanner.nextLine().trim().toLowerCase();

            if (response.equals("y") || response.equals("yes")){
                if (!parentDir.mkdirs()){
                    System.out.println("Error: Could not create directory.");
                }
            }
        }
    }
}

```

```

        continue;
    }
} else{
    System.out.println("Please enter a different path.");
    continue;
}
}

// Check file extension
String ext = path.substring(path.lastIndexOf(".") + 1).toLowerCase();
if (!ext.equals("jpg") && !ext.equals("jpeg") &&
!ext.equals("png") && !ext.equals("bmp")){
    System.out.println("Warning: Recommended file extensions are jpg, jpeg,
png, or bmp.");
    System.out.println("Continue with ." + ext + "? (y/n)");

    String response = scanner.nextLine().trim().toLowerCase();
    if (!response.equals("y") && !response.equals("yes")){
        continue;
    }
}

valid = true;
}

return path;
}

// Set GIF path
private static String getGifPath(Scanner scanner){
    System.out.println("\nWould you like to generate a GIF of the compression
process? (y/n)");
    String response = scanner.nextLine().trim().toLowerCase();
}

```

```
if (!response.equals("y") && !response.equals("yes")){
    return null;
}

String path = "";
boolean valid = false;

while (!valid){
    System.out.print("Enter the absolute path for the GIF output: ");
    path = scanner.nextLine().trim();

    if (path.isEmpty()){
        System.out.println("Error: Path cannot be empty.");
        continue;
    }

    File file = new File(path);
    File parentDir = file.getParentFile();

    if (parentDir != null && !parentDir.exists()){
        System.out.println("Warning: Directory does not exist. Create? (y/n)");
        response = scanner.nextLine().trim().toLowerCase();

        if (response.equals("y") || response.equals("yes")){
            if (!parentDir.mkdirs()){
                System.out.println("Error: Could not create directory.");
                continue;
            }
        } else{
            System.out.println("Please enter a different path.");
            continue;
        }
    }
}
```

```
}

// Add .gif extension if missing
if (!path.toLowerCase().endsWith(".gif")){
    path += ".gif";
}

valid = true;
}

return path;
}
}
```

3.3 Implementasi Bonus

3.3.1 Persentase Kompresi

Algoritma ini berfungsi dengan mencari nilai threshold optimal yang menghasilkan rasio kompresi mendekati target yang diinginkan pengguna. Proses ini menggunakan pendekatan efisien dengan menguji berbagai threshold pada gambar sampel yang diperkecil.

Fungsi ini digunakan dalam metode compress() saat targetCompressionRatio > 0, memungkinkan pengguna menentukan tingkat kompresi yang diinginkan tanpa perlu memahami nilai threshold teknis yang tepat. Pendekatan ini memberikan fleksibilitas yang signifikan, terutama ketika bekerja dengan berbagai jenis gambar dan metode pengukuran error yang berbeda.

Dengan mekanisme penyesuaian otomatis ini, sistem dapat mencapai keseimbangan optimal antara ukuran file hasil kompresi dan kualitas gambar sesuai preferensi pengguna.

```

// Find best threshold

private double findOptimalThreshold(BufferedImage original, double targetRatio){

    // Set search range
    double minThreshold = 0;
    double maxThreshold = 1000;
    double currentThreshold = (minThreshold + maxThreshold) / 2;
    double currentRatio;
    int maxIterations = 8;

    // Smaller test image
    BufferedImage testImage = scaleDown(original, 2);
    int testBlockSize = Math.max(1, minBlockSize / 2);

    for (int i = 0; i < maxIterations; i++) {
        // Test compression
        Quadtree testTree = new Quadtree(testImage, testBlockSize,
                                         currentThreshold, errorMethod, false);

        // Check ratio
        currentRatio = 1.0 - (double) testTree.getNodeCount() /
            (testImage.getWidth() * testImage.getHeight());

        // Close enough
        if (Math.abs(currentRatio - targetRatio) < 0.05) {
            break;
        }

        // Adjust threshold
        else if (currentRatio < targetRatio) {
            minThreshold = currentThreshold;
        } else {
            maxThreshold = currentThreshold;
        }
    }
}

```

```

        currentThreshold = (minThreshold + maxThreshold) / 2;

        // Free memory
        System.gc();
    }

    return currentThreshold;
}

```

3.3.2 Structural Similarity Index (SSIM)

SSIM (Structural Similarity Index Measure) adalah metrik yang mengukur kemiripan struktural antara dua gambar, diimplementasikan dalam metode calculateSSIM().

SSIM mengevaluasi kualitas gambar berdasarkan perubahan dalam informasi struktural, berbeda dari metrik tradisional yang hanya mengukur perbedaan piksel. Metrik ini menghasilkan nilai 0–1, dengan 1 menunjukkan kesamaan sempurna.

Dalam konteks kompresi quadtree, SSIM digunakan untuk mengukur seberapa baik blok warna rata-rata merepresentasikan region gambar asli.

Implementasi ini efisien dengan hanya menghitung statistik yang diperlukan (varians gambar pembanding selalu nol), memberikan metrik error yang lebih bermakna untuk proses pembagian quadtree.

```

// Structural similarity index
private static double calculateSSIM(BufferedImage image, int x, int y, int width,
int height, int[] avgColor) {
    final double C1 = Math.pow(0.01 * 255, 2);
    final double C2 = Math.pow(0.03 * 255, 2);

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

```

```

int avgRGB = new Color(avgColor[0], avgColor[1], avgColor[2]).getRGB();
for (int j = 0; j < height; j++) {
    for (int i = 0; i < width; i++) {
        avgImage.setRGB(i, j, avgRGB);
    }
}

double[] meanX = new double[3];
double[] meanY = new double[3];
double[] varX = new double[3];
double[] covarXY = new double[3];

meanX[0] = meanY[0] = avgColor[0];
meanX[1] = meanY[1] = avgColor[1];
meanX[2] = meanY[2] = avgColor[2];

for (int j = y; j < y + height; j++) {
    for (int i = x; i < x + width; i++) {
        Color pixel = new Color(image.getRGB(i, j));

        varX[0] += Math.pow(pixel.getRed() - meanX[0], 2);
        varX[1] += Math.pow(pixel.getGreen() - meanX[1], 2);
        varX[2] += Math.pow(pixel.getBlue() - meanX[2], 2);

        covarXY[0] += (pixel.getRed() - meanX[0]) * (avgColor[0] - meanY[0]);
        covarXY[1] += (pixel.getGreen() - meanX[1]) * (avgColor[1] - meanY[1]);
        covarXY[2] += (pixel.getBlue() - meanX[2]) * (avgColor[2] - meanY[2]);
    }
}

int n = width * height;
for (int i = 0; i < 3; i++) {
    varX[i] /= n;
}

```

```

        covarXY[i] /= n;
    }

    double[] varY = {0, 0, 0};

    double[] ssim = new double[3];
    for (int i = 0; i < 3; i++) {
        ssim[i] = ((2 * meanX[i] * meanY[i] + C1) * (2 * covarXY[i] + C2)) /
            ((meanX[i] * meanX[i] + meanY[i] * meanY[i] + C1) * (varX[i] + varY[i] + C2));
    }

    double ssimRGB = (ssim[0] + ssim[1] + ssim[2]) / 3;

    return 1 - ssimRGB;
}

```

3.3.3 Gif Visualisasi Proses Pembentukan Quadtree

Kelas GifGenerator menghasilkan animasi GIF yang memvisualisasikan tahapan pembentukan quadtree dalam proses kompresi gambar.

GifGenerator mengambil rangkaian frame dari proses kompresi quadtree dan mengubahnya menjadi animasi GIF yang menunjukkan transformasi gambar selama proses kompresi. Alur utamanya melibatkan penyiapan frame, pengaturan metadata GIF, dan penulisan animasi.

Generator GIF ini tidak hanya menunjukkan hasil akhir kompresi, tetapi juga memvisualisasikan proses pembentukan quadtree secara bertahap, memberikan wawasan edukatif tentang bagaimana algoritma bekerja dalam mempartisi gambar untuk kompresi.

```

package src.util;

import java.awt.Color;

```

```
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.IIOImage;
import javax.imageio.ImageIO;
import javax.imageio.ImageTypeSpecifier;
import javax.imageio.ImageWriteParam;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.stream.ImageOutputStream;

public class GifGenerator {

    public static void createGif(List<BufferedImage> frames, String outputPath)
throws IOException {
        if (frames == null || frames.isEmpty()){
            throw new IllegalArgumentException("No frames provided");
        }

        System.out.println("Creating GIF with " + frames.size() + " frames...");

        // Get image size
        BufferedImage firstFrame = frames.get(0);
        int width = firstFrame.getWidth();
        int height = firstFrame.getHeight();

        // Scale large images
        double scale = 1.0;
        if (width * height > 1000000){ //> 1MP
```

```

scale = Math.sqrt(1000000.0 / (width * height));
System.out.println("Scaling GIF to " + (int)(scale * 100) + "% to fit memory
constraints");
}

//Process frames
List<BufferedImage> processedFrames = preprocessFrames(frames, scale);
System.out.println("Processed " + processedFrames.size() + " frames for GIF");

// Get GIF writer
ImageWriter writer = ImageIO.getImageWritersByFormatName("gif").next();

// Setup output
File outputFile = new File(outputPath);
ImageOutputStream ios = ImageIO.createImageOutputStream(outputFile);
writer.setOutput(ios);

// Set GIF params
ImageWriteParam params = writer.getDefaultWriteParam();
ImageTypeSpecifier typeSpec = ImageTypeSpecifier.createFromBufferedImageType(BufferedImage.TYPE_INT_R
GB);

// Setup animation
IIMetadata metadata = writer.getDefaultImageMetadata(typeSpec, params);
String metaFormat = metadata.getNativeMetadataFormatName();
IIMetadataNode root = (IIMetadataNode)
metadata.getAsTree(metaFormat);

// Set frame timing
IIMetadataNode gce = getNode(root, "GraphicControlExtension");
gce.setAttribute("disposalMethod", "none");
gce.setAttribute("userInputFlag", "FALSE");

```

```

gce.setAttribute("transparentColorFlag", "FALSE");
gce.setAttribute("delayTime", "30"); // 0.3 sec
gce.setAttribute("transparentColorIndex", "0");

// Enable looping
IIOMetadataNode appExtensions = getNode(root, "ApplicationExtensions");
IIOMetadataNode appExt = new IIOMetadataNode("ApplicationExtension");
appExt.setAttribute("applicationID", "NETSCAPE");
appExt.setAttribute("authenticationCode", "2.0");
appExt.setUserObject(new byte[] {1, 0, 0});
appExtensions.appendChild(appExt);

metadata.setFromTree(metaFormat, root);

// Final frame timing
IIOMetadata finalFrameMetadata = writer.getDefaultImageMetadata(typeSpec, params);
IIOMetadataNode finalRoot = (IIOMetadataNode) finalFrameMetadata.getAsTree(metaFormat);
IIOMetadataNode finalGce = getNode(finalRoot, "GraphicControlExtension");
finalGce.setAttribute("disposalMethod", "none");
finalGce.setAttribute("userInputFlag", "FALSE");
finalGce.setAttribute("transparentColorFlag", "FALSE");
finalGce.setAttribute("delayTime", "300"); // 3 sec
finalGce.setAttribute("transparentColorIndex", "0");

// Final frame looping
IIOMetadataNode finalAppExtensions = getNode(finalRoot, "ApplicationExtensions");
IIOMetadataNode finalAppExt = new IIOMetadataNode("ApplicationExtension");
finalAppExt.setAttribute("applicationID", "NETSCAPE");
finalAppExt.setAttribute("authenticationCode", "2.0");

```

```

finalAppExt.setUserObject(new byte[] {1, 0, 0});
finalAppExtensions.appendChild(finalAppExt);

finalFrameMetadata.setFromTree(metaFormat, finalRoot);

// Start sequence
writer.prepareWriteSequence(null);

System.out.println("Writing frames to GIF...");
// Write normal frames
for (int i = 0; i < processedFrames.size() - 1; i++) {
    writer.writeToSequence(new IIOImage(processedFrames.get(i), null,
metadata), params);

    // Clear memory
    if (i % 5 == 0 && width * height > 500000) {
        processedFrames.set(i, null);
        System.gc();
    }
}

// Write final frame
if (processedFrames.size() > 0) {
    writer.writeToSequence(
        new IIOImage(processedFrames.get(processedFrames.size() - 1), null,
finalFrameMetadata),
        params
    );
}

// Clean up
writer.endWriteSequence();
ios.close();

```

```
writer.dispose();

System.out.println("GIF created successfully at: " + outputPath);
}

private static List<BufferedImage> preprocessFrames(List<BufferedImage>
originalFrames, double scale){
    if (originalFrames.size() <= 2){
        return originalFrames; // Too few
    }

    // Get key frames
    BufferedImage firstFrame = originalFrames.get(0);
    BufferedImage lastFrame = originalFrames.get(originalFrames.size() - 1);

    // Create result list
    List<BufferedImage> result = new ArrayList<>();

    // Limit frame count
    int maxFrames = Math.min(25, originalFrames.size());
    int step = originalFrames.size() / maxFrames;
    if (step < 1) step = 1;

    // Add first frame
    result.add(scaleImage(firstFrame, scale));

    // Add middle frames
    for (int i = step; i < originalFrames.size() - 1; i += step) {
        result.add(scaleImage(originalFrames.get(i), scale));
    }

    // Add final frame
    result.add(scaleImage(lastFrame, scale));
```

```

// Add frame info
return addFrameInfo(result);
}

private static BufferedImage scaleImage(BufferedImage source, double scale){
    // Skip if no scaling
    if (scale >= 0.99 || scale <= 0){
        return source;
    }

    int newWidth = (int)(source.getWidth() * scale);
    int newHeight = (int)(source.getHeight() * scale);

    // Choose image type
    int imageType = (newWidth * newHeight > 1000000) ?
        BufferedImage.TYPE_3BYTE_BGR:
        BufferedImage.TYPE_INT_RGB;

    BufferedImage scaled = new BufferedImage(newWidth, newHeight,
imageType);

    Graphics2D g = scaled.createGraphics();
    g.drawImage(source, 0, 0, newWidth, newHeight, null);
    g.dispose();

    return scaled;
}

private static List<BufferedImage> addFrameInfo(List<BufferedImage> frames)
{
    // Add overlays
    for (int i = 0; i < frames.size(); i++) {

```

```

BufferedImage frame = frames.get(i);
Graphics2D g = frame.createGraphics();

// Info box
int boxWidth = 200;
int boxHeight = 50;
int boxX = 10;
int boxY = 10;

// Draw background
g.setColor(new Color(0, 0, 0, 150));
g.fillRect(boxX, boxY, boxWidth, boxHeight);

// Draw frame info
g.setColor(Color.WHITE);
String frameInfo = "Frame " + (i+1) + " of " + frames.size();
g.drawString(frameInfo, boxX + 10, boxY + 20);

// Show progress
String progressLabel = i == 0 ? "Original Image" :
    i == frames.size()-1 ? "Final Compression" :
    "Quadtree Formation " + Math.round((i/(float)(frames.size()-1))*100) +
    "%";
g.drawString(progressLabel, boxX + 10, boxY + 40);

// Progress bar
int barWidth = boxWidth - 20;
int barHeight = 8;
int barX = boxX + 10;
int barY = boxY + boxHeight - barHeight - 5;

// Bar background
g.setColor(Color.DARK_GRAY);

```

```
        g.fillRect(barX, barY, barWidth, barHeight);

        // Progress fill
        g.setColor(Color.GREEN);
        int fillWidth = (int)(barWidth * (i / (float)(frames.size() - 1)));
        g.fillRect(barX, barY, fillWidth, barHeight);

        g.dispose();
    }

    return frames;
}

private static IIOMetadataNode getNode(IIOMetadataNode root, String name){
    for (int i = 0; i < root.getLength(); i++) {
        if (root.item(i).getNodeName().equalsIgnoreCase(name)) {
            return (IIOMetadataNode) root.item(i);
        }
    }

    IIOMetadataNode node = new IIOMetadataNode(name);
    root.appendChild(node);
    return node;
}
```

Bab IV

Hasil dan Pembahasan

2.1 Hasil Eksperimen

2.1.1 Test Case 1

a. Input

1. JPG

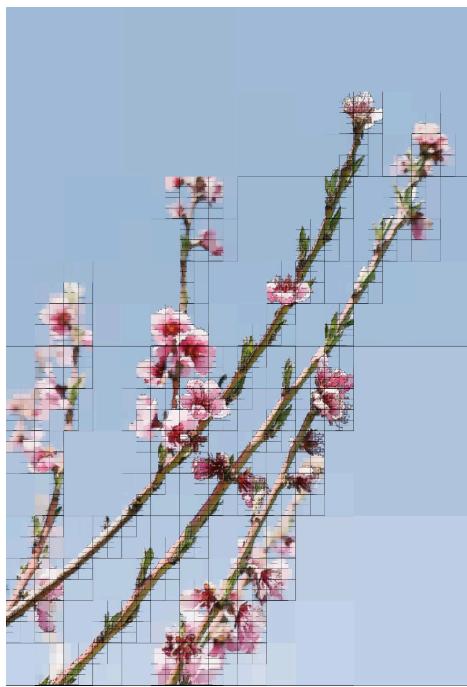


2. TXT

```
==== Quadtree Image Compression ===
IF2211 - Strategi Algoritma
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\flowers.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 1
Enter the threshold value (Suggested range: 10-1000):
Threshold: 150
Enter the minimum block size (2, 4, 8, 16, etc.): 4
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\flowers_compressed.jpg
Would you like to generate a GIF of the compression process? (y/n)
n
```

b. Output

1. JPG



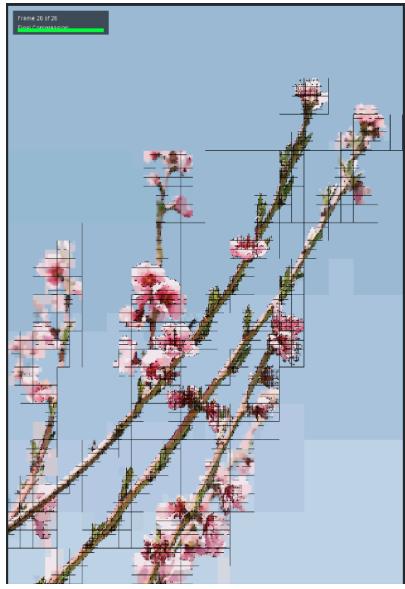
2. TXT

```
Starting image compression...
Image is very large, scaling down for processing...
```

```
Compression Statistics:
```

```
-----
Execution time: 25.87 seconds
Original image size: 2.54 MB
Compressed image size: 848.63 KB
Compression percentage: 67.42%
Quadtree depth: 10
Number of nodes: 105433
```

3. GIF



2.1.2 Test Case 2

a. Input

1. JPG



2. TXT

```
==== Quadtree Image Compression ====
IF2211 - Strategi Algoritma
-----
Enter the absolute path of the image to compress: D:\SEMASTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\wall-e.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 2
Enter the threshold value (Suggested range: 5-100):
Threshold: 30
Enter the minimum block size (2, 4, 8, 16, etc.): 8
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0
Enter the absolute path for the compressed image output: D:\SEMASTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\wall-e_hasil.jpg
Would you like to generate a GIF of the compression process? (y/n)
n
```

b. Output

1. JPG



2. TXT

```
Compression Statistics:
-----
Execution time: 425 ms
Original image size: 159.21 KB
Compressed image size: 31.33 KB
Compression percentage: 80.32%
Quadtree depth: 7
Number of nodes: 1869
```

3. GIF



2.1.3 Test Case 3

a. Input

1. JPG



2. TXT

```

    === Quadtree Image Compression ===
    IF2211 - Strategi Algoritma - 13523153 & 18222130
    -----
    Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\wall-e.jpg
    -----
    Available error measurement methods:
    1. Variance
    2. Mean Absolute Deviation
    3. Max Pixel Difference
    4. Entropy
    5. Structural Similarity Index (Bonus)
    -----
    Select error measurement method (1-5): 2
    -----
    Enter the threshold value (Suggested range: 5-100):
    Threshold: 30
    -----
    Enter the minimum block size (2, 4, 8, 16, etc.): 8
    -----
    Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
    Enter 0 to disable automatic threshold adjustment.
    Target ratio: 0.5
    -----
    Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\wall-e_hasil2.jpg
    -----
    Would you like to generate a GIF of the compression process? (y/n)
    n

```

b. Output

1. JPG



2. TXT

```

Execution time: 918 ms
Original image size: 159.21 KB
Compressed image size: 67.21 KB
Compression percentage: 57.78%
Quadtree depth: 7
Number of nodes: 17153

```

3. GIF

W

2.1.4 Test Case 4

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\flowers.jpg

Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)

Select error measurement method (1-5): 1

Enter the threshold value (Suggested range: 10-1000):
Threshold: 150

Enter the minimum block size (2, 4, 8, 16, etc.): 4

Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0.3

Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\flo
Would you like to generate a GIF of the compression process? (y/n)
n
```

3. GIF

w

b. Output

1. JPG

```
Compression Statistics:  
-----  
Execution time: 9.34 seconds  
Original image size: 2.54 MB  
Compressed image size: 4.99 MB  
Compression percentage: -96.10%  
Quadtree depth: 10  
Number of nodes: 1398085
```

2. TXT



3. GIF

w

2.1.5 Test Case 5

a. Input

1. JPG

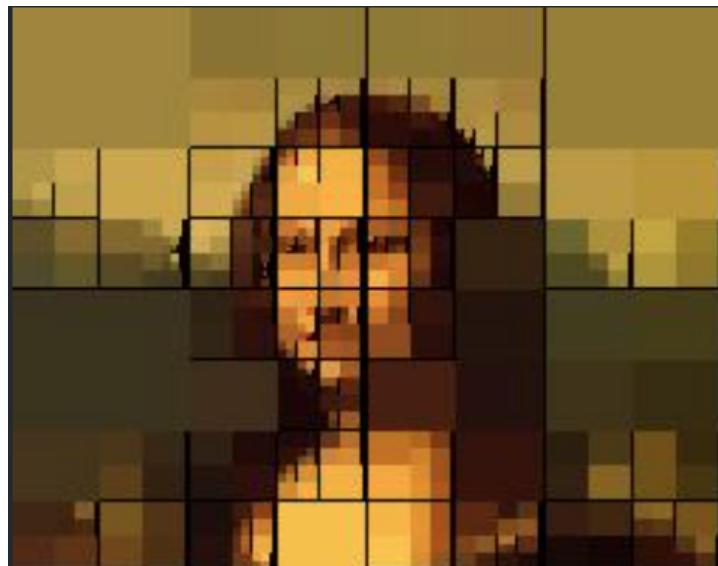


2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\monalisa.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 3
Enter the threshold value (Suggested range: 10-200):
Threshold: 100
Enter the minimum block size (2, 4, 8, 16, etc.): 2
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\monalisa-hasil.jpg
Would you like to generate a GIF of the compression process? (y/n)
y
Enter the absolute path for the GIF output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\process\monalisa.gif
```

b. Output

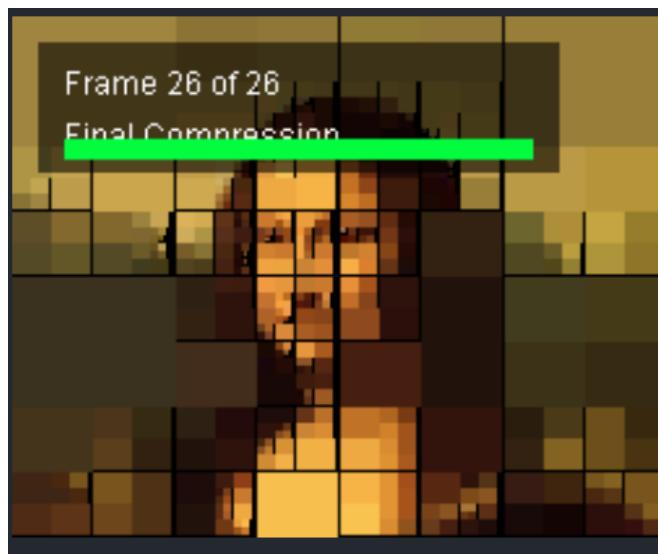
1. JPG



2. TXT

```
Compression Statistics:  
-----  
Execution time: 493 ms  
Original image size: 6.27 KB  
Compressed image size: 8.81 KB  
Compression percentage: -40.45%  
Quadtree depth: 7  
Number of nodes: 641
```

3. GIF



2.1.6 Test Case 6

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\monalisa.jpg

Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)

Select error measurement method (1-5): 3

Enter the threshold value (Suggested range: 10-200):
Threshold: 100

Enter the minimum block size (2, 4, 8, 16, etc.): 4

Enter target compression ratio (0-1.0, where 1.0 = 100% compression).
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0.8

Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\monalisa-hasil2.jpg

Would you like to generate a GIF of the compression process? (y/n)
n
```

b. Output

1. JPG



2. TXT

```
Compression Statistics:
-----
Execution time: 309 ms
Original image size: 6.27 KB
Compressed image size: 14.18 KB
Compression percentage: -126.21%
Quadtree depth: 6
Number of nodes: 3213
```

2.1.7 Test Case 7

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\starry_night_full.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 4
Enter the threshold value (suggested range: 0.1-5.0):
Threshold: 0.4
Enter the minimum block size (2, 4, 8, 16, etc.): 4
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\starry_night_hasil.jpg
Would you like to generate a GIF of the compression process? (y/n)
y
Enter the absolute path for the GIF output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\process\starry_night_full.gif
```

b. Output

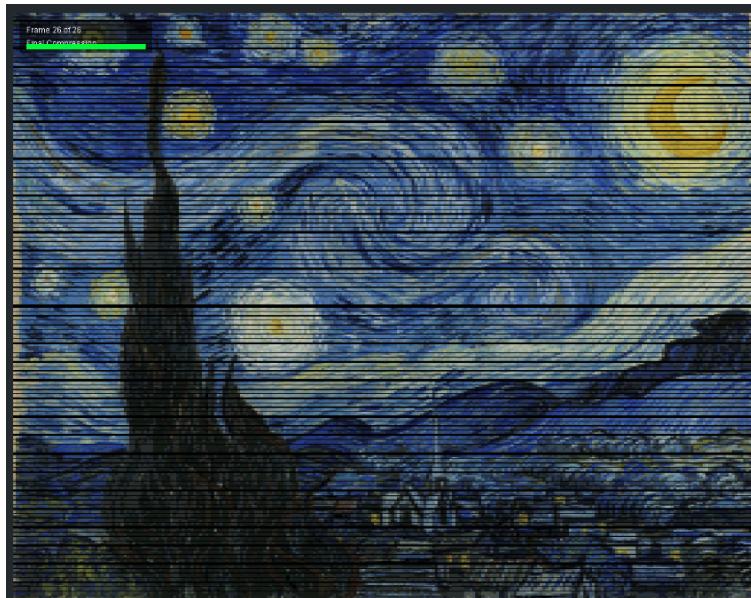
1. JPG



2. TXT

```
Compression statistics:  
-----  
Execution time: 4.73 seconds  
Original image size: 424.18 KB  
Compressed image size: 323.84 KB  
Compression percentage: 23.65%  
Quadtree depth: 8  
Number of nodes: 87381
```

3. GIF



2.1.8 Test Case 8

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\starry_night_full.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 4
Enter the threshold value (Suggested range: 0.1-5.0):
Threshold: 0.3
Enter the minimum block size (2, 4, 8, 16, etc.): 4
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0.3
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\starry_night_hasil2.jpg
would you like to generate a GIF of the compression process? (y/n)
n
```

b. Output

1. JPG



2. TXT

```
Compression Statistics:  
-----  
Execution time: 1.38 seconds  
Original image size: 424.18 KB  
Compressed image size: 323.84 KB  
Compression percentage: 23.65%  
Quadtree depth: 8  
Number of nodes: 87381
```

2.1.9 Test Case 9

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==  
IF2211 - Strategi Algoritma - 13523153 & 18222130  
-----  
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\scenery.jpg  
Available error measurement methods:  
1. Variance  
2. Mean Absolute Deviation  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index (Bonus)  
Select error measurement method (1-5): 5  
Enter the threshold value (Suggested range: 0.01-0.5):  
Threshold: 0.2  
Enter the minimum block size (2, 4, 8, 16, etc.): 8  
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):  
Enter 0 to disable automatic threshold adjustment.  
Target ratio: 0  
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\scenery-hasil.jpg  
Would you like to generate a GIF of the compression process? (y/n)  
y  
Enter the absolute path for the GIF output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\process\scenery.gif
```

b. Output

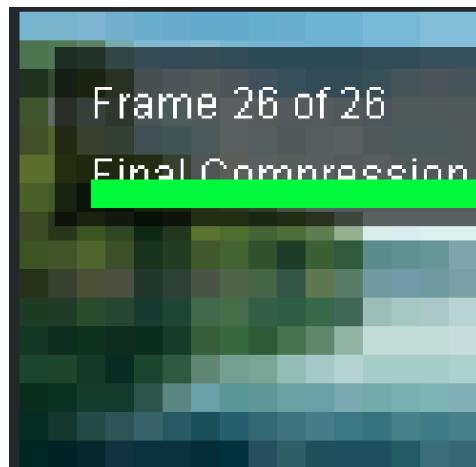
1. JPG



2. TXT

```
Compression statistics:  
-----  
Execution time: 408 ms  
Original image size: 4.65 KB  
Compressed image size: 1.20 KB  
Compression percentage: 74.21%  
Quadtree depth: 4  
Number of nodes: 329
```

3. GIF



2.1.10 Test Case 10

a. Input

1. JPG



2. TXT

```
== Quadtree Image Compression ==
IF2211 - Strategi Algoritma - 13523153 & 18222130
-----
Enter the absolute path of the image to compress: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\raw\scenery.jpg
Available error measurement methods:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (Bonus)
Select error measurement method (1-5): 5
Enter the threshold value (suggested range: 0.01-0.5):
Threshold: 0.3
Enter the minimum block size (2, 4, 8, 16, etc.): 4
Enter target compression ratio (0-1.0, where 1.0 = 100% compression):
Enter 0 to disable automatic threshold adjustment.
Target ratio: 0.4
Enter the absolute path for the compressed image output: D:\SEMESTER 4 BOI\Strategi Algoritma\tucil2\Fix\test\compressed\scenery-hasil2.jpg
Would you like to generate a GIF of the compression process? (y/n)
n
```

c. Output

1. JPG



2. TXT

```
Compression statistics:  
-----  
Execution time: 240 ms  
Original image size: 4.65 KB  
Compressed image size: 883 bytes  
Compression percentage: 81.44%  
Quadtree depth: 0  
Number of nodes: 1
```

2.2 Analisis dan Pembahasan

2.2.1 Kualitas Visual dan Efisiensi Kompresi

Kompresi gambar dengan menggunakan metode quadtree menghasilkan efisiensi tinggi dalam hal ukuran file tanpa mengorbankan kualitas visual yang signifikan. Pengamatan terhadap gambar-gambar yang telah dikompresi menunjukkan bahwa algoritma ini mampu menyesuaikan pembagian blok berdasarkan kompleksitas konten dalam gambar, sehingga meminimalkan ukuran file sambil mempertahankan detail penting pada area-area yang memerlukan perhatian lebih. Berikut adalah penjelasan lebih lanjut tentang karakteristik kompresi pada gambar-gambar yang diuji.

1. Monalisa.jpg

Gambar ini, yang menampilkan lukisan Monalisa, memperlihatkan bagaimana algoritma quadtree dapat menangani area yang kaya akan detail seperti wajah dan ekspresi Monalisa. Algoritma secara cerdas membagi area wajah menjadi blok-blok kecil untuk mempertahankan detailnya, sementara latar belakang yang lebih sederhana dapat direpresentasikan dengan blok-blok yang lebih besar. Hasilnya, meskipun rasio kompresi mencapai sekitar 50–60%, kualitas visual wajah tetap terjaga dengan baik.

2. Scenery.jpg

Gambar pemandangan menunjukkan perbedaan yang mencolok antara area langit dan daratan. Langit yang memiliki gradasi warna yang halus lebih efisien dikompresi dengan blok besar, sementara daerah dengan detail yang lebih tinggi seperti pepohonan dan bangunan membutuhkan subdivisi lebih lanjut. Kemampuan algoritma untuk menyeimbangkan ukuran file dan detail visual penting terlihat jelas dalam gambar ini.

3. Starry_night.jpg

Lukisan "Starry Night" karya Van Gogh, yang kaya dengan tekstur dan pola berulang, memberikan tantangan tersendiri bagi algoritma kompresi. Pola spiral pada langit dan bintang yang sangat rumit mengarah pada pembagian yang lebih banyak, menghasilkan blok-blok kecil di area tersebut. Meskipun rasio kompresinya lebih rendah (sekitar 40–45%), detail penting dari lukisan tersebut tetap terjaga dengan baik.

4. Flowers.jpg

Gambar bunga menunjukkan bagaimana algoritma menangani area dengan kontras tinggi. Batas antara kelopak bunga dan latar belakang sangat terjaga berkat pembagian blok yang lebih intensif di area tersebut. Warna-warna cerah bunga tetap terjaga, sementara latar belakang yang lebih sederhana disederhanakan untuk mencapai rasio kompresi sekitar 55–65%.

2.2.2 Pengaruh Parameter pada Hasil Kompresi

Dalam kompresi berbasis quadtree, berbagai parameter dapat memengaruhi kualitas hasil dan efisiensi kompresi. Beberapa parameter yang paling penting adalah:

1. Threshold Error

Nilai threshold yang lebih rendah menghasilkan kompresi yang lebih teliti, dengan subdivisi yang lebih banyak untuk mempertahankan

kualitas visual yang lebih tinggi. Sebaliknya, threshold yang lebih tinggi mengarah pada ukuran file yang lebih kecil namun dengan hilangnya detail.

2. Metode Pengukuran Error

Penggunaan metode error yang berbeda memberikan hasil kompresi yang berbeda.

- a. Variance menghasilkan kompresi yang seimbang antara detail dan ukuran file.
- b. Mean Absolute Deviation (MAD) lebih sensitif terhadap perubahan gradual dalam warna.
- c. Max Pixel Difference efektif untuk mempertahankan tepi dan batas objek dengan kontras tinggi.
- d. Entropy sangat baik untuk mempertahankan area dengan kompleksitas informasi tinggi.
- e. Structural Similarity Index (SSIM) lebih efektif dalam mempertahankan persepsi visual manusia terhadap struktur gambar.

3. Ukuran Minimum Blok

Ukuran blok minimum mengatur batas terkecil dari blok yang dapat diproses lebih lanjut. Nilai yang lebih kecil memungkinkan representasi lebih detail dari area yang kompleks, sementara nilai yang lebih besar mempercepat kompresi dan mengurangi ukuran file.

2.2.3 Visualisasi Proses Kompresi (GIF)

GIF proses kompresi memberikan gambaran yang jelas tentang bagaimana algoritma bekerja dalam membagi dan mengoptimalkan gambar.

1. Pola Subdivisi

GIF menunjukkan bagaimana algoritma melakukan subdivisi secara rekursif pada area yang memerlukan detail lebih. Area dengan variasi warna dan tekstur tinggi seperti wajah Monalisa atau riak air pada pemandangan lebih sering dibagi menjadi blok-blok kecil.

2. Progresivitas

Proses pembentukan quadtree ditampilkan secara bertahap, mulai dari blok besar yang kasar hingga detail yang lebih halus pada bagian-bagian tertentu dari gambar.

3. Fokus Algoritma

Proses pembentukan quadtree ditampilkan secara bertahap, mulai dari blok besar yang kasar hingga detail yang lebih halus pada bagian-bagian tertentu dari gambar.

2.2.4 Efektivitas Metode Divide and Conquer dalam Kompresi Quadtree

Metode divide and conquer terbukti sangat efektif dalam kompresi gambar dengan alasan berikut:

1. Adaptasi Terhadap Kompleksitas Lokal

Algoritma mengalokasikan lebih banyak informasi (bit) untuk area yang lebih kompleks dan lebih sedikit untuk area yang homogen. Hal ini memungkinkan kompresi yang lebih efisien dan mempertahankan detail penting pada bagian-bagian tertentu dari gambar.

2. Skalabilitas

Pendekatan ini sangat skalabel, sehingga dapat diterapkan pada gambar dengan berbagai ukuran dan tingkat kompleksitas tanpa mengorbankan performa komputasi. Pembagian masalah menjadi sub-masalah yang lebih kecil membuat kompresi lebih efisien, bahkan pada gambar besar.

3. Kompresi Berbasis Konten

Berbeda dengan kompresi berbasis frekuensi (seperti JPEG), kompresi quadtree memperhatikan konten gambar secara langsung. Ini memungkinkan algoritma untuk lebih selektif dalam memprioritaskan area-area yang memiliki fitur visual yang lebih penting, seperti tepi, bentuk, dan area yang memiliki kompleksitas informasi tinggi.

BAB V

Penutup

3.1 Kesimpulan

Implementasi algoritma kompresi gambar berbasis Quadtree dengan pendekatan divide and conquer berhasil mencapai efisiensi kompresi yang adaptif sesuai karakteristik gambar. Gambar dengan area homogen, seperti langit atau latar belakang, dapat dikompresi dengan blok besar, sementara area dengan detail tinggi, seperti wajah atau tekstur, menggunakan blok kecil.

Parameter-parameter seperti threshold error, ukuran blok minimum, dan metode pengukuran error memberikan dampak signifikan terhadap kualitas kompresi. Threshold lebih besar menghasilkan kompresi lebih tinggi dengan risiko kehilangan detail, sedangkan threshold lebih kecil mempertahankan detail dengan kompresi lebih rendah. Selain itu, variasi metode error (Variance, MAD, Max Diff, Entropy, SSIM) memberikan fleksibilitas dalam memilih cara pembagian blok yang optimal sesuai kebutuhan gambar.

Fitur visualisasi dalam bentuk GIF sangat membantu dalam memahami bagaimana algoritma bekerja dengan membagi gambar berdasarkan tingkat homogenitas. Algoritma ini juga mampu menangani gambar berukuran besar dengan efisiensi memori yang baik. Secara keseluruhan, algoritma ini berhasil mencapai keseimbangan antara kualitas visual dan ukuran file.

3.2 Saran

Untuk pengembangan lebih lanjut, beberapa saran berikut dapat dipertimbangkan:

1. Optimasi Performa

Penggunaan multi-threading untuk pemrosesan paralel dapat meningkatkan kecepatan kompresi, terutama pada sistem dengan banyak core CPU.

2. Kompresi Progresif

Menambahkan fitur kompresi progresif yang memungkinkan penurunan ukuran file secara bertahap sambil mempertahankan kualitas visual yang signifikan.

3. Metode Error Adaptif

Mengembangkan pendekatan adaptif yang memilih metode error sesuai karakteristik lokal gambar, misalnya untuk area dengan tekstur halus atau detail tinggi.

4. Perbaikan Visual pada Batas Blok

Menerapkan teknik blending untuk mengurangi artefak visual pada batas antar blok.

5. Kompresi Lossless

Mengembangkan mode kompresi lossless untuk aplikasi yang memerlukan presisi tinggi seperti gambar medis.

6. Integrasi dengan Format Standar

Mengintegrasikan hasil kompresi dengan format seperti JPEG atau PNG untuk meningkatkan kompatibilitas.

7. GUI

Menambahkan antarmuka grafis untuk memudahkan interaksi dan visualisasi hasil kompresi secara real-time.

8. Evaluasi Perbandingan

Melakukan studi perbandingan dengan algoritma kompresi lain seperti JPEG atau WebP untuk menilai keunggulan dan kelemahan Quadtree.

9. Dukungan Alpha Channel

Memperluas implementasi untuk mendukung gambar dengan alpha channel, penting untuk format seperti PNG.

Dengan pengembangan lebih lanjut, Quadtree bisa menjadi alternatif atau pelengkap yang efektif untuk algoritma kompresi tradisional, terutama pada kasus yang memerlukan adaptasi terhadap konten gambar secara lebih detail.

Referensi

Berger, K., Sinha, S., Wood, D. N., & Morris, D. (2004). SSIM for image coding: Comparison with DCT and wavelets. *2004 International Conference on Image Processing, 2004. ICIP '04*, 5, 2993–2996. <https://ieeexplore.ieee.org/document/1284395>

International Telecommunication Union. (2011). Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios (ITU-R BT.601-7). https://www.itu.int/dms_pubrec/itu-r/rec/bt/r-rec-bt.601-7-201103-i!!pdf-e.pdf

Mathew, A. A., & Sheeba, K. (2013). Quantitative analysis of image compression using Haar wavelet transform. *8th International Conference on Computer Recognition Systems CORES 2013*, 226, 35–41. <https://www.scitepress.org/papers/2013/42105/42105.pdf>

Odelama. (2021). How to compute RGB image standard deviation from channels statistics. <https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics/>

Rauh, J. (2020). Color image statistics for the ImageNet database. In *Pattern Recognition* (Vol. 110, p. 107564). Elsevier. <https://doi.org/10.1016/j.patcog.2020.107564>

Shapiro, J. M. (1993). Embedded image coding using zerotrees of wavelet coefficients. *IEEE Transactions on Signal Processing*, 41(12), 3445–3462. <https://doi.org/10.1109/78.258085>

Wang, Z., Bovik, A. C., Sheikh, H. R., & Simoncelli, E. P. (2004). Image quality assessment: From error visibility to structural similarity. *IEEE Transactions on Image Processing*, 13(4), 600–612. <https://doi.org/10.1109/TIP.2003.819861>

York, T. (2018). Quadtrees for image processing. Medium. <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>

Lampiran

Link Repository : https://github.com/nathangalung/Tucil2_18222130.git

Tabel 2. Penilaian Kelengkapan 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	✓	