

IF2211 – Strategi Algoritma

Kompresi Gambar dengan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2 Tahun Akademik 2024/2025



Disusun oleh:

Theo Kurniady (13523154)
Filbert Engyo (13523163)

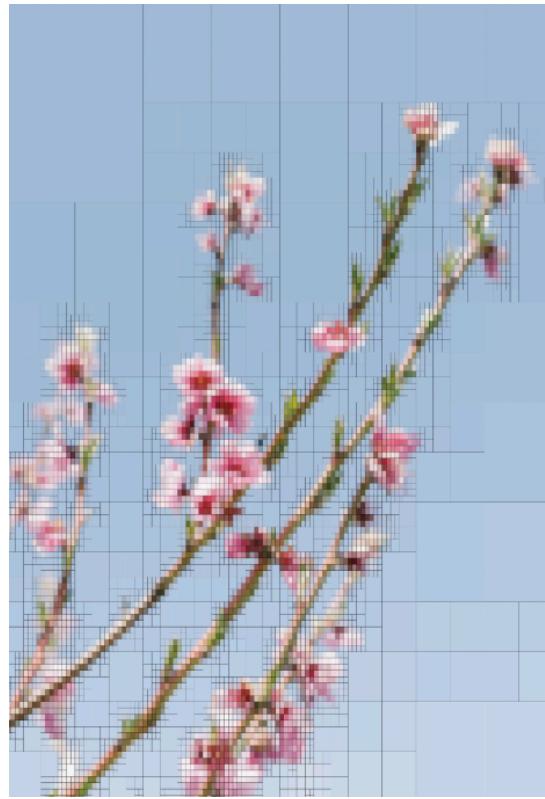
**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132
2024**

Daftar Isi

Laporan Tugas Kecil 2.....	1
Daftar Isi.....	2
BAB I	
DESKRIPSI MASALAH.....	3
BAB II.....	5
DASAR TEORI.....	5
2.1 Divide and Conquer.....	5
2.2 QuadTree.....	6
2.3 Image Quality Assessment.....	7
2.3.1 Variance.....	7
2.3.2 Mean Absolute Deviation.....	8
2.3.3 Max Pixel Difference.....	8
2.3.4 Entropy.....	9
2.3.5. Structural Similarity Index (SSIM).....	9
2.4. Teorema Master.....	10
BAB III	
ALGORITMA PEMROGRAMAN.....	11
BAB IV	
IMPLEMENTASI PROGRAM.....	13
4. 1. QuadTreeNode.....	14
4.2. CompressionUtils.....	15
4.3. Compression (main).....	19
BAB V	
EKSPERIMENT & ANALISIS.....	24
LAMPIRAN.....	30
DAFTAR PUSTAKA.....	31

BAB I

DESKRIPSI MASALAH



Gambar 1. Quadtree dalam Kompresi Gambar
(Sumber:

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

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

Dalam implementasi teknis, sebuah *QuadTree* direpresentasikan sebagai simpul (*node*) dengan maksimal empat anak (*children*). Simpul daun (*leaf*) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi

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

Tugas kali ini meminta kami untuk membuat program sederhana dalam bahasa C/C#/C++/Java (CLI) yang mengimplementasikan algoritma divide and conquer untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan seluruh parameter yang telah disebutkan sebagai user input.

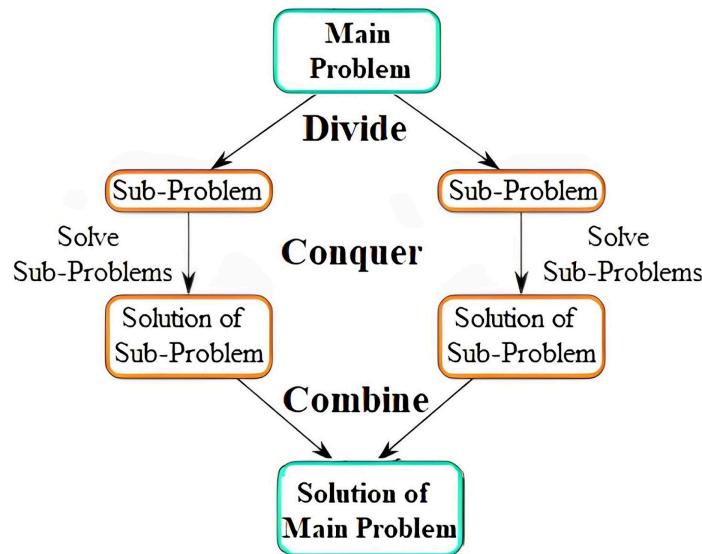
Alur program:

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] alamat absolut gambar hasil kompresi.
6. [OUTPUT] waktu eksekusi.
7. [OUTPUT] ukuran gambar sebelum.
8. [OUTPUT] ukuran gambar setelah.
9. [OUTPUT] persentase kompresi.
10. [OUTPUT] kedalaman pohon.
11. [OUTPUT] banyak simpul pada pohon.
12. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.

BAB II

DASAR TEORI

2.1 Divide and Conquer



Gambar 2.1 Divide & Conquer

(Sumber : [Data Structures and Algorithms Cheat Sheet + PDF | Zero To Mastery](#))

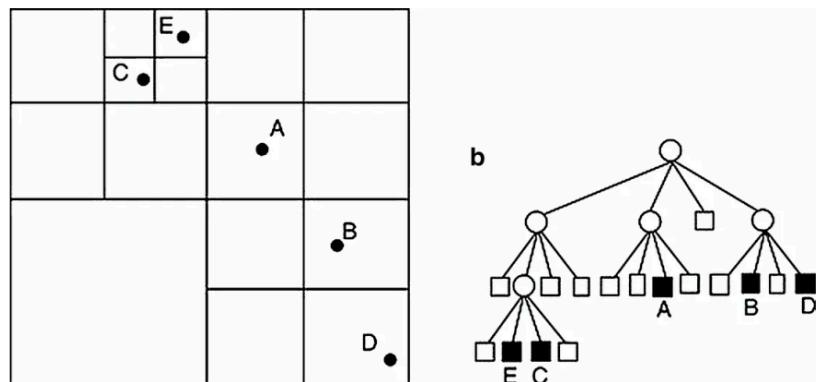
Divide and Conquer adalah paradigma algoritma yang menyelesaikan suatu permasalahan dengan membaginya menjadi sub-masalah yang lebih kecil, menyelesaikan masing-masing sub-masalah secara rekursif, lalu menggabungkan hasilnya untuk membentuk solusi akhir. Pendekatan ini umumnya terdiri dari tiga tahap utama:

1. **Divide** – Pada tahap ini, masalah utama dibagi menjadi dua atau lebih sub-masalah yang berukuran lebih kecil namun memiliki sifat atau struktur yang sama dengan masalah asal. Proses pembagian ini dilakukan secara sistematis agar ukuran sub-masalah terus mengecil pada tiap langkah rekursi. Contohnya, dalam algoritma *Merge Sort*, daftar data dibagi menjadi dua bagian yang kira-kira sama besar. Dalam konteks pemrosesan citra, gambar besar bisa dibagi menjadi blok-blok kecil untuk analisis lokal, misalnya berdasarkan keseragaman warna atau intensitas piksel. Tujuan dari tahap ini adalah mengurangi kompleksitas masalah besar menjadi masalah-masalah yang lebih mudah dikelola. Ukuran pemisahan bisa tetap (misalnya 2 bagian), atau dinamis tergantung kompleksitas data.

2. **Conquer** - Setelah masalah dibagi, setiap sub-masalah diselesaikan secara rekursif. Jika sub-masalah masih terlalu besar, maka proses divide akan dilakukan lagi, membentuk struktur rekursif berlapis hingga mencapai base case, yaitu kondisi di mana masalah cukup kecil sehingga dapat diselesaikan secara langsung (tanpa perlu dibagi lagi). Tujuan dari tahap ini adalah menyelesaikan semua bagian kecil yang terbentuk dari proses pembagian sebelumnya.
3. **Merge** – Setelah semua sub-masalah diselesaikan, tahap terakhir adalah menggabungkan hasil-hasilnya menjadi satu solusi utuh dari masalah utama. Cara penggabungan ini sangat tergantung pada jenis permasalahan yang dihadapi. Dalam *QuadTree*, blok-blok homogen yang berdekatan bisa digabung jika memenuhi kriteria tertentu, atau hasil evaluasi tiap blok disatukan untuk keperluan analisis global.

2.2 *QuadTree*

QuadTree adalah struktur data hierarkis yang digunakan untuk merepresentasikan data dua dimensi dengan cara melakukan rekursif dalam membagi ruang menjadi empat bagian kuadran yang sama besar. Setiap simpul (*node*) dalam *QuadTree* dapat memiliki nol atau empat anak. Struktur ini sangat berguna dalam pengolahan citra, kompresi gambar, indeks spasial, dan segmentasi citra, karena mampu merepresentasikan area gambar yang homogen secara efisien.



Gambar 2.2 Struktur data quadtree dalam ruang 2D

(Sumber :

https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm)

Pada dasarnya, proses pembentukan struktur *QuadTree* dimulai dengan memeriksa apakah suatu blok dalam citra bersifat homogen, yaitu seluruh pikselnya memiliki nilai

yang relatif sama. Jika ya, maka blok tersebut dianggap sebagai simpul daun dan tidak dibagi lebih lanjut. Namun, jika blok mengandung variasi intensitas atau warna yang signifikan, maka blok tersebut akan dibagi menjadi empat bagian yang sama besar dan proses yang sama dilakukan kembali pada masing-masing bagian. Proses ini dilakukan secara rekursif hingga seluruh area gambar terbagi menjadi blok-blok homogen atau mencapai batas pembagian tertentu. Dengan demikian, *QuadTree* secara otomatis menyesuaikan tingkat detail pembagian sesuai dengan kompleksitas konten dalam gambar.

2.3 *Image Quality Assessment*

Image Quality Assessment (IQA) atau penilaian kualitas citra adalah proses untuk mengukur seberapa baik suatu citra digital mewakili atau mempertahankan informasi visual, baik dibandingkan dengan citra asli (referensi) maupun tanpa referensi. IQA penting dalam banyak bidang seperti kompresi gambar, transmisi video, pemrosesan citra medis, serta pengembangan algoritma peningkatan kualitas visual. Penilaian ini dapat dilakukan secara subjektif, yaitu berdasarkan persepsi manusia, atau secara objektif menggunakan metode-metode matematis yang mengkuantifikasi kualitas gambar berdasarkan metrik tertentu. Penilaian objektif lebih umum digunakan dalam sistem otomatis karena bersifat konsisten dan dapat diukur secara terprogram.

Penilaian kualitas citra secara objektif sering kali menggunakan metrik statistik yang mengukur variasi dan distribusi nilai piksel dalam gambar. Beberapa metrik yang akan digunakan adalah variance, mean absolute deviation, max pixel difference, dan entropy.

2.3.1 *Variance*

Variance (varian) adalah metrik statistik yang mengukur tingkat penyebaran nilai piksel terhadap nilai rata-ratanya. Dalam konteks citra digital, varian digunakan untuk mengetahui seberapa besar perbedaan antar piksel dalam suatu blok gambar. Nilai varian yang tinggi menunjukkan bahwa terdapat banyak perubahan intensitas dalam blok tersebut, yang bisa berarti detail atau noise, tergantung konteksnya. Sebaliknya, nilai varian yang rendah mengindikasikan bahwa nilai-nilai piksel saling mendekati, mencerminkan area gambar yang homogen. Berikut adalah rumus dari varian.

$$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$$

- σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
 $P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
 μ_c = Nilai rata-rata tiap piksel dalam satu blok
 N = Banyaknya piksel dalam satu blok

2.3.2 Mean Absolute Deviation

Mean Absolute Deviation (MAD) adalah metrik statistik yang juga digunakan untuk mengukur sebaran nilai piksel terhadap nilai rata-rata, namun berbeda dengan varian, MAD menghitung rata-rata dari nilai absolut selisih antara tiap piksel dan nilai rata-rata intensitas. MAD cenderung lebih stabil dan tidak terlalu terpengaruh oleh nilai-nilai ekstrem (outlier) seperti varian. Oleh karena itu, MAD sering dianggap sebagai alternatif yang lebih robust dalam mengukur penyebaran piksel, terutama pada citra dengan noise atau artefak. Berikut adalah rumus dari MAD.

$$MAD_c = \frac{1}{N} \sum_{i=1}^N |P_{i,c} - \mu_c|$$

- MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B)
dalam satu blok
 $P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
 μ_c = Nilai rata-rata tiap piksel dalam satu blok
 N = Banyaknya piksel dalam satu blok

2.3.3 Max Pixel Difference

Max Pixel Difference adalah metrik sederhana yang mengukur perbedaan maksimum antara nilai piksel tertinggi dan terendah dalam sebuah blok citra. Metrik ini memberikan informasi cepat tentang kontras lokal dalam blok tersebut. Jika selisih antara nilai tertinggi dan terendah sangat besar, berarti ada variasi tajam dalam area tersebut — bisa berupa perbedaan warna mendadak, tepi objek, atau bahkan noise. Berikut rumus dari Max Pixel Difference.

$$D_c = \max(P_{i,c}) - \min(P_{i,c})$$

D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c

2.3.4 Entropy

Entropy dalam konteks citra digital merujuk pada tingkat ketidakpastian atau keragaman informasi dalam sebuah blok gambar. Konsep ini berasal dari teori informasi, di mana entropy mengukur jumlah rata-rata informasi yang dibutuhkan untuk merepresentasikan isi dari suatu sinyal — dalam hal ini, sinyal visual berupa nilai piksel. *Entropy* tinggi berarti nilai-nilai piksel tersebar secara merata di banyak level intensitas, menunjukkan keberagaman dan detail yang tinggi. Sebaliknya, *entropy* rendah menunjukkan bahwa piksel-piksel cenderung memiliki nilai yang serupa, atau informasi dalam blok tersebut sangat terbatas. Berikut adalah rumus dari *entropy*:

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok

$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

2.3.5. Structural Similarity Index (SSIM)

Structural Similarity Index (SSIM) adalah sebuah metrik yang digunakan untuk mengukur kesamaan struktural antara dua gambar atau dua bagian gambar. SSIM mengukur tiga aspek utama: *luminance* (pencahayaan), kontras, dan struktur. Dengan membandingkan komponen-komponen ini, SSIM dapat memberikan nilai yang menunjukkan seberapa mirip dua gambar secara visual. Nilai SSIM berkisar antara -1 hingga 1, di mana 1 menunjukkan kesamaan sempurna antara gambar-gambar tersebut. Metrik ini sering digunakan dalam evaluasi kualitas gambar dan kompresi karena lebih mencerminkan persepsi manusia terhadap kualitas gambar dibandingkan dengan metrik seperti MSE (*Mean Squared Error*) atau PSNR (*Peak Signal-to-Noise Ratio*).

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

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

2.4. Teorema Master

Teorema Master adalah teknik yang digunakan untuk menyelesaikan persamaan rekursif untuk banyak algoritma *divide and conquer* yang memiliki bentuk umum:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + O(n)$$

Dengan keterangan:

- N : ukuran masalah
- A : jumlah pemanggilan rekursif pada setiap langkah
- n/b : faktor pengurangan ukuran masalah dalam setiap pemanggilan rekursif
- O(n) : pekerjaan tambahan yang dilakukan di luar pemanggilan rekursif.

Teorema Master membagi masalah ke dalam tiga kasus berdasarkan hubungan antara a, b, dan d:

1. Kasus 1

Jika $a < b^d$, maka kompleksitas waktu adalah $O(n^d)$

2. Kasus 2

Jika $a = b^d$, maka kompleksitas waktu adalah $O(n^d \log n)$

3. Kasus 3

Jika $a > b^d$, maka kompleksitas waktu adalah $O(n^{\lceil \log_b a \rceil})$

BAB III

ALGORITMA PEMROGRAMAN

Program dibagi menjadi beberapa tahap dengan masing-masing tujuan yang berbeda. Input program berupa parameter yang telah dijelaskan di bab sebelumnya. Setelah melakukan input gambar ke program, dimulai proses kompresi gambar dengan algoritma *divide and conquer*. Berikut adalah tahapan dari algoritma *divide and conquer*

3.1. *Divide*

Tahap pertama dari algoritma adalah *divide*, yaitu saat program mulai memecah citra menjadi empat bagian kuadran yang berukuran relatif seimbang. Setiap blok hasil pembagian disebut sebagai sub-gambar atau sub-blok. Proses ini dilakukan secara rekursif dan terus berlanjut selama blok saat ini dianggap belum cukup sederhana untuk dikompresi. Tujuan utama dari proses ini adalah untuk mengidentifikasi area-area dalam gambar yang lebih homogen pada skala lokal, sehingga mereka dapat ditangani lebih mudah pada tahap selanjutnya. Semakin dalam (*depth*) pembagian sub-blok, semakin kecil ukuran blok, dan semakin spesifik masalah kompresi yang ingin diselesaikan.

3.2. *Conquer*

Setelah blok-blok kecil diperoleh dari tahap pembagian, program mulai menyelesaikan setiap blok. Dalam konteks ini, prosesnya melibatkan perhitungan tingkat ketidakhomogenan atau kesalahan (error) dalam masing-masing blok menggunakan metrik IQA (*Image Quality Assessment*) seperti *variance*, *mean absolute deviation*, *max pixel difference*, atau *entropy* sesuai pilihan pengguna. Jika nilai error suatu blok melebihi ambang batas yang ditentukan dan ukuran blok masih cukup besar untuk dibagi, maka blok tersebut akan kembali dibagi. Namun, jika blok sudah memenuhi syarat homogen atau ukurannya terlalu kecil untuk dibagi lagi (sebesar min blok size), maka blok tersebut dianggap selesai. Blok ini kemudian akan direpresentasikan dengan satu nilai warna yang dinormalisasi (misalnya, rata-rata nilai intensitas), sebagai solusi lokal dari permasalahan kompresi di area tersebut.

3.3. *Merge*

Setelah semua proses rekursif selesai dan tiap blok homogen telah ditentukan, program memasuki tahap terakhir, yaitu penggabungan (merge). Di sini, hasil dari setiap blok terminal atau *leaf node* dalam struktur *QuadTree* akan digabungkan kembali untuk membentuk representasi citra hasil kompresi. Penggabungan akan menghasilkan gambar baru yang memiliki ukuran data lebih kecil, namun tetap menjaga informasi visual yang relevan, bila diatur dengan baik. Gambar akhir ini adalah bentuk terkompresi dari gambar

asli, yang dihasilkan dari penyelesaian masalah kompresi melalui pendekatan *divide and conquer*.

BAB IV

IMPLEMENTASI PROGRAM

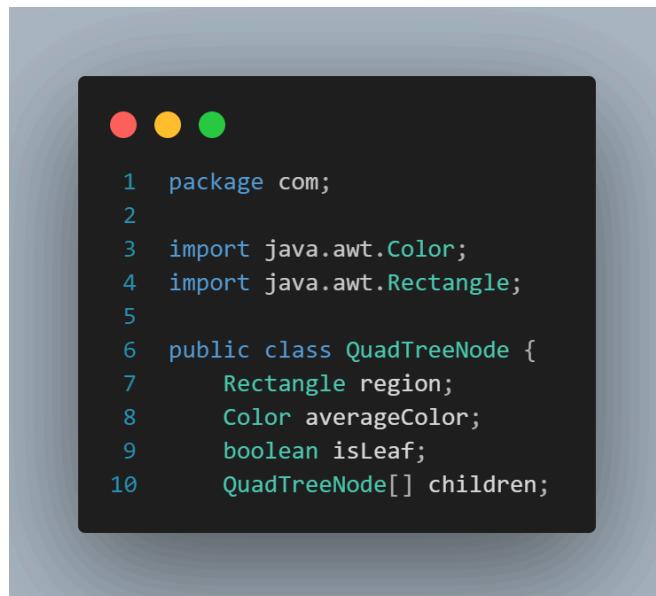
Program diimplementasikan dalam bahasa pemrograman Java yang memanfaatkan paradigma objek. Untuk memudahkan proses penggunaan, segala file yang dibuat dimasukkan dalam *package* com. Secara garis besar, struktur folder yang dibuat seperti dibawah:

```
Tucil2_13523154_13523163/
├── bin/
│   ├── Compression.class
│   ├── CompressionUtils.class
│   └── QuadTreeNode.class
├── doc/
│   └── Tucil2_K3_13523154_13523160
└── src/
    └── com/
        ├── Compression.java
        ├── CompressionUtils.java
        └── QuadTreeNode.java
└── test/
    └── output.png
└── {build.bat, build.sh, README.md}
```

File main atau driver utama adalah *Compression.java*, didalamnya terdapat fungsi-fungsi pembantu untuk program utama. Untuk pemrosesan secara garis besar terdapat pada *CompressionUtil.java*, dan untuk proses implementasi algoritma *divide and conquer* diterapkan pada pengolahan *tree* yang dibuat dalam *QuadTreeNode.java*.

4. 1. *QuadTreeNode*

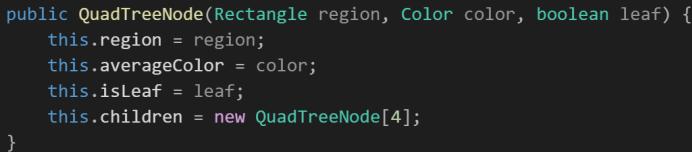
Kelas *QuadTreeNode* memiliki struktur data yang khusus mencakup tipe data rectangle yang menjadi region atau blok yang membagi gambar, data color untuk warna rata-rata sebuah blok, sebuah boolean *isLeaf* untuk mengidentifikasi *leaf (no children)*, dan array *QuadTreeNode* yang berupa children hasil pembagian 4 sub-blok:



```
1 package com;
2
3 import java.awt.Color;
4 import java.awt.Rectangle;
5
6 public class QuadTreeNode {
7     Rectangle region;
8     Color averageColor;
9     boolean isLeaf;
10    QuadTreeNode[] children;
```

Gambar 4.1.1. Struktur Data Kelas *QuadTreeNode*

Berdasarkan struktur data dibuatlah fungsi inisialisasi atau *constructor* yang menginisialisasi pembuatan tree sesuai kebutuhan berupa:



```
1 public QuadTreeNode(Rectangle region, Color color, boolean leaf) {
2     this.region = region;
3     this.averageColor = color;
4     this.isLeaf = leaf;
5     this.children = new QuadTreeNode[4];
6 }
```

Gambar 4.1.2. Konstruktor *QuadTreeNode*

Berdasarkan konstruktor yang ada, dibuatlah fungsi pembantu yaitu untuk menghitung *node* dan kedalaman *QuadTreeNode* yang nanti akan membantu pembuatan *QuadTreeNode*, fungsi yang dibuat sebagai berikut:



```
1 public static int countDepths(QuadTreeNode node) {
2     if (node == null || node.isLeaf) return 0;
3     int maxDepth = 0;
4     for (QuadTreeNode child : node.children) {
5         maxDepth = Math.max(maxDepth, countDepths(child));
6     }
7     return maxDepth + 1;
8 }
9
10 public static int countNodes(QuadTreeNode node) {
11     if (node == null) return 0;
12     int count = 1;
13     for (QuadTreeNode child : node.children) {
14         count += countNodes(child);
15     }
16     return count;
17 }
```

Gambar 4.1.3. Fungsi *countDepths* & *countNodes*

4.2. *CompressionUtils*

Kelas *CompressionUtils* berisi fungsi-fungsi pembantu yang melakukan operasi terhadap kalkulasi kompresi gambar yang memanfaatkan struktur *QuadTreeNode*. Fungsi-fungsi ini berfokus pada berbagai cara yang digunakan untuk mengevaluasi wilayah gambar dan menentukan cara mengompres atau menganalisis kualitas gambar. Fungsi pembantu dari fungsi opsional ada 3 yaitu *averageColor* untuk menghitung rata-rata warna wilayah dari suatu gambar, *drawCompressedImage* untuk membentuk kompresi gambar berdasarkan struktur *QuadTreeNode* yang telah terbentuk, dan *entropyHelper* untuk membantu perhitungan metode entropy dalam menghitung histogram:



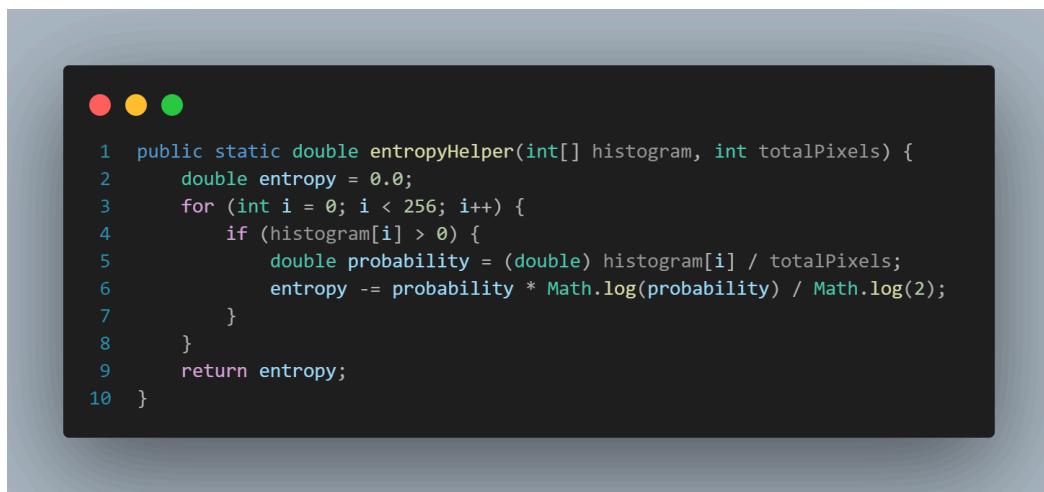
```
1 public static Color AverageColor(BufferedImage img, Rectangle region) {
2     int sumR = 0, sumG = 0, sumB = 0, count = 0;
3     for (int y = region.y; y < region.y + region.height; y++) {
4         for (int x = region.x; x < region.x + region.width; x++) {
5             Color c = new Color(img.getRGB(x, y));
6             sumR += c.getRed();
7             sumG += c.getGreen();
8             sumB += c.getBlue();
9             count++;
10        }
11    }
12    return new Color(sumR / count, sumG / count, sumB / count);
13 }
```

Gambar 4.2.1. Fungsi *AverageColor*



```
1 public static void drawCompressedImage(Graphics g, QuadTreeNode node) {
2     if (node == null) return;
3
4     Graphics2D g2d = (Graphics2D) g;
5     g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING, RenderingHints.VALUE_ANTIALIAS_OFF);
6
7     if (node.isLeaf) {
8         g2d.setColor(node.averageColor);
9         g2d.fillRect(node.region.x, node.region.y, node.region.width, node.region.height);
10    } else {
11        for (QuadTreeNode child : node.children) {
12            drawCompressedImage(g2d, child);
13        }
14    }
15 }
```

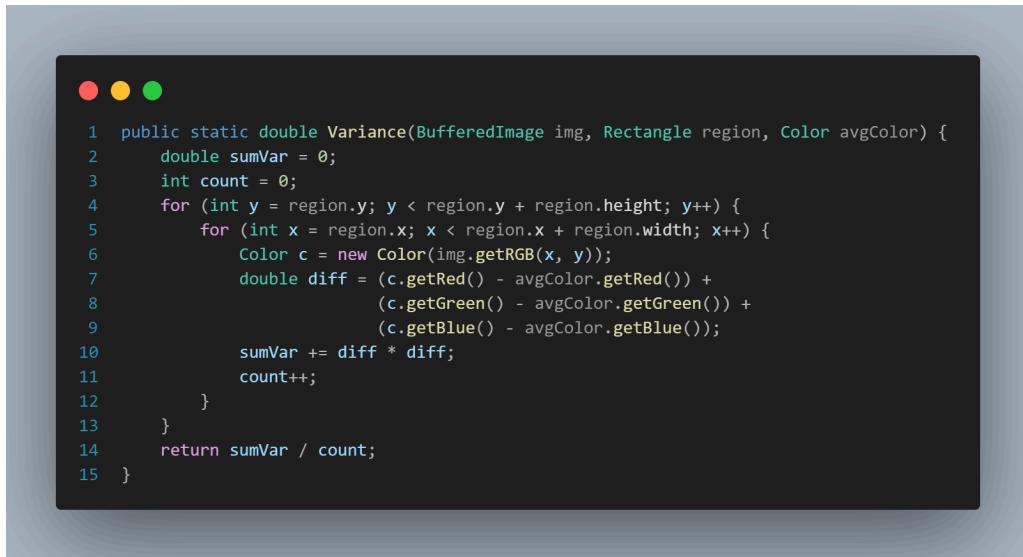
Gambar 4.2.2. Fungsi *drawCompressedImage*



```
1 public static double entropyHelper(int[] histogram, int totalPixels) {
2     double entropy = 0.0;
3     for (int i = 0; i < 256; i++) {
4         if (histogram[i] > 0) {
5             double probability = (double) histogram[i] / totalPixels;
6             entropy -= probability * Math.log(probability) / Math.log(2);
7         }
8     }
9     return entropy;
10 }
```

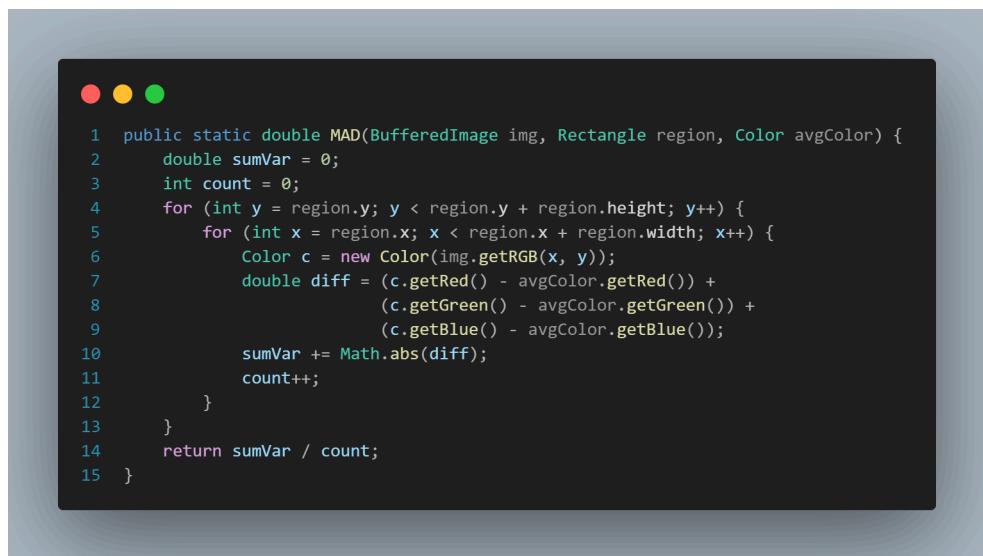
Gambar 4.2.3. Fungsi *entropyHelper*

Berdasarkan itu, dibuatlah fungsi *image quality assessment* yang merupakan opsi metode untuk menghitung pengukuran kesalahan. Metode itu terdiri dari *variance*, *mean absolute deviation* (MAD), *max pixel difference*, *entropy* yang implementasi perhitungannya seperti dibawah ini:



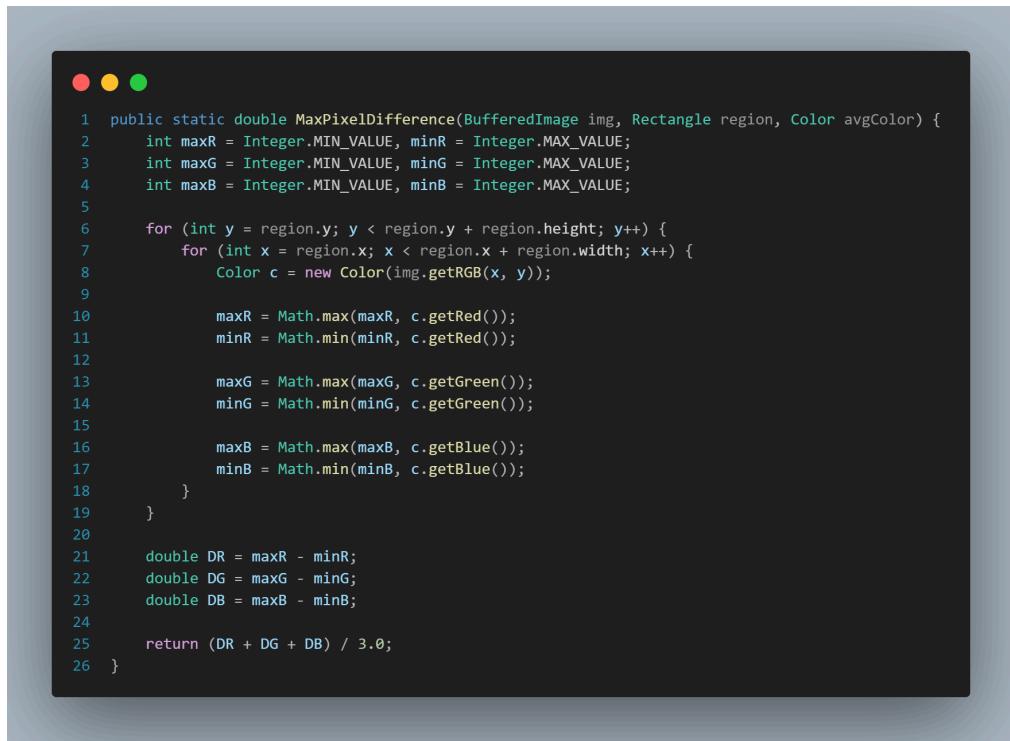
```
1 public static double Variance(BufferedImage img, Rectangle region, Color avgColor) {  
2     double sumVar = 0;  
3     int count = 0;  
4     for (int y = region.y; y < region.y + region.height; y++) {  
5         for (int x = region.x; x < region.x + region.width; x++) {  
6             Color c = new Color(img.getRGB(x, y));  
7             double diff = (c.getRed() - avgColor.getRed()) +  
8                         (c.getGreen() - avgColor.getGreen()) +  
9                         (c.getBlue() - avgColor.getBlue());  
10            sumVar += diff * diff;  
11            count++;  
12        }  
13    }  
14    return sumVar / count;  
15 }
```

Gambar 4.2.4. Fungsi Metode *Variance*



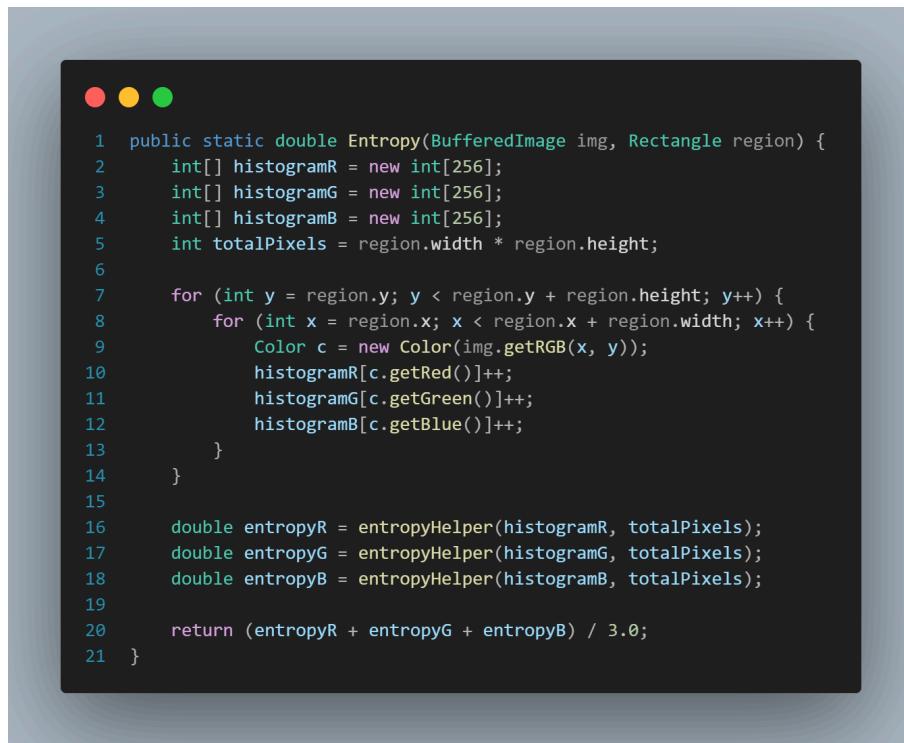
```
1 public static double MAD(BufferedImage img, Rectangle region, Color avgColor) {  
2     double sumVar = 0;  
3     int count = 0;  
4     for (int y = region.y; y < region.y + region.height; y++) {  
5         for (int x = region.x; x < region.x + region.width; x++) {  
6             Color c = new Color(img.getRGB(x, y));  
7             double diff = (c.getRed() - avgColor.getRed()) +  
8                         (c.getGreen() - avgColor.getGreen()) +  
9                         (c.getBlue() - avgColor.getBlue());  
10            sumVar += Math.abs(diff);  
11            count++;  
12        }  
13    }  
14    return sumVar / count;  
15 }
```

Gambar 4.2.5. Fungsi Metode *Mean Absolute Deviation* (MAD)



```
1 public static double MaxPixelDifference(BufferedImage img, Rectangle region, Color avgColor) {
2     int maxR = Integer.MIN_VALUE, minR = Integer.MAX_VALUE;
3     int maxG = Integer.MIN_VALUE, minG = Integer.MAX_VALUE;
4     int maxB = Integer.MIN_VALUE, minB = Integer.MAX_VALUE;
5
6     for (int y = region.y; y < region.y + region.height; y++) {
7         for (int x = region.x; x < region.x + region.width; x++) {
8             Color c = new Color(img.getRGB(x, y));
9
10            maxR = Math.max(maxR, c.getRed());
11            minR = Math.min(minR, c.getRed());
12
13            maxG = Math.max(maxG, c.getGreen());
14            minG = Math.min(minG, c.getGreen());
15
16            maxB = Math.max(maxB, c.getBlue());
17            minB = Math.min(minB, c.getBlue());
18        }
19    }
20
21    double DR = maxR - minR;
22    double DG = maxG - minG;
23    double DB = maxB - minB;
24
25    return (DR + DG + DB) / 3.0;
26 }
```

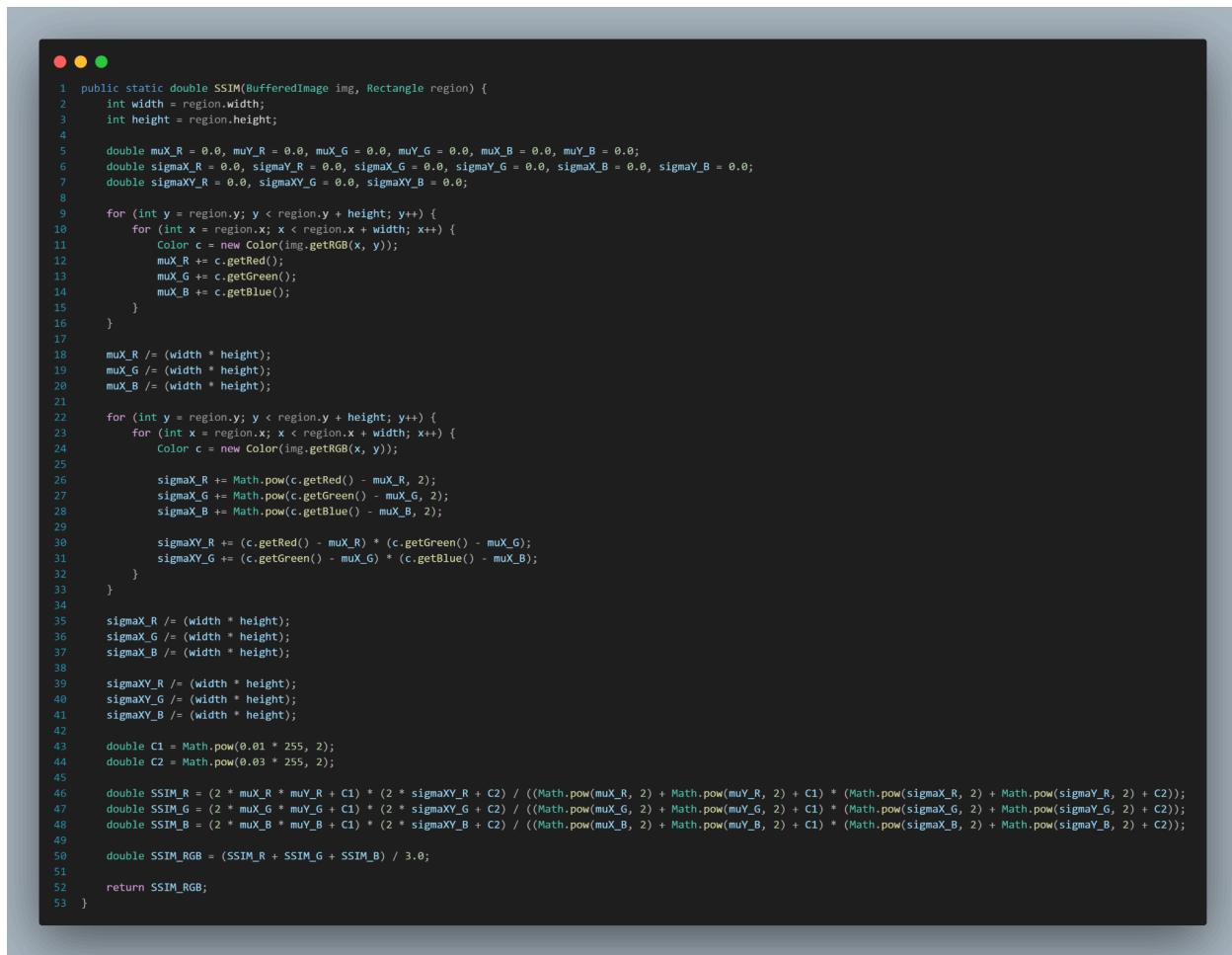
Gambar 4.2.6. Fungsi Metode Max Pixel Difference



```
1 public static double Entropy(BufferedImage img, Rectangle region) {
2     int[] histogramR = new int[256];
3     int[] histogramG = new int[256];
4     int[] histogramB = new int[256];
5     int totalPixels = region.width * region.height;
6
7     for (int y = region.y; y < region.y + region.height; y++) {
8         for (int x = region.x; x < region.x + region.width; x++) {
9             Color c = new Color(img.getRGB(x, y));
10            histogramR[c.getRed()]++;
11            histogramG[c.getGreen()]++;
12            histogramB[c.getBlue()]++;
13        }
14    }
15
16    double entropyR = entropyHelper(histogramR, totalPixels);
17    double entropyG = entropyHelper(histogramG, totalPixels);
18    double entropyB = entropyHelper(histogramB, totalPixels);
19
20    return (entropyR + entropyG + entropyB) / 3.0;
21 }
```

Gambar 4.2.7. Fungsi Metode *Entropy*

Pada program ini, diimplementasikan bonus yaitu metode pengukuran error menggunakan SSIM yang prosesnya diawali dengan penghitungan rata-rata dan variansi untuk setiap warna (R, G, B). Lalu dihitung kovariansi dari tiap dua warna yang berbeda yaitu antara R dengan G dan B dengan B. Berdasarkan data yang diperoleh, dilakukan perhitungan menggunakan rumus SSIM untuk masing-masing warna dengan menambahkan konstanta kecil (C1 dan C2) untuk mencegah pembagian oleh nol. SSIM dihitung untuk setiap warna, lalu nilai akhirnya dihitung sebagai rata-rata dari SSIM R, G, dan B sehingga membentuk nilai SSIM RGB yang mencerminkan kesamaan struktural antara wilayah gambar yang dianalisis:



```

1  public static double SSIM(BufferedImage img, Rectangle region) {
2      int width = region.width;
3      int height = region.height;
4
5      double muX_R = 0.0, muY_R = 0.0, muX_G = 0.0, muY_G = 0.0, muX_B = 0.0, muY_B = 0.0;
6      double sigmaX_R = 0.0, sigmaY_R = 0.0, sigmaX_G = 0.0, sigmaY_G = 0.0, sigmaX_B = 0.0, sigmaY_B = 0.0;
7      double sigmaXY_R = 0.0, sigmaXY_G = 0.0, sigmaXY_B = 0.0;
8
9      for (int y = region.y; y < region.y + height; y++) {
10          for (int x = region.x; x < region.x + width; x++) {
11              Color c = new Color(img.getRGB(x, y));
12              muX_R += c.getRed();
13              muX_G += c.getGreen();
14              muX_B += c.getBlue();
15          }
16      }
17      muX_R /= (width * height);
18      muX_G /= (width * height);
19      muX_B /= (width * height);
20
21      for (int y = region.y; y < region.y + height; y++) {
22          for (int x = region.x; x < region.x + width; x++) {
23              Color c = new Color(img.getRGB(x, y));
24
25              sigmaX_R += Math.pow(c.getRed() - muX_R, 2);
26              sigmaX_G += Math.pow(c.getGreen() - muX_G, 2);
27              sigmaX_B += Math.pow(c.getBlue() - muX_B, 2);
28
29              sigmaXY_R += (c.getRed() - muX_R) * (c.getGreen() - muX_G);
30              sigmaXY_G += (c.getGreen() - muX_G) * (c.getBlue() - muX_B);
31              sigmaXY_B += (c.getBlue() - muX_B) * (c.getRed() - muX_R);
32          }
33      }
34
35      sigmaX_R /= (width * height);
36      sigmaX_G /= (width * height);
37      sigmaX_B /= (width * height);
38
39      sigmaXY_R /= (width * height);
40      sigmaXY_G /= (width * height);
41      sigmaXY_B /= (width * height);
42
43      double C1 = Math.pow(0.01 * 255, 2);
44      double C2 = Math.pow(0.03 * 255, 2);
45
46      double SSIM_R = (2 * muX_R * muY_R + C1) * (2 * sigmaXY_R + C2) / ((Math.pow(muX_R, 2) + Math.pow(muY_R, 2) + C1) * (Math.pow(sigmaX_R, 2) + Math.pow(sigmaY_R, 2) + C2));
47      double SSIM_G = (2 * muX_G * muY_G + C1) * (2 * sigmaXY_G + C2) / ((Math.pow(muX_G, 2) + Math.pow(muY_G, 2) + C1) * (Math.pow(sigmaX_G, 2) + Math.pow(sigmaY_G, 2) + C2));
48      double SSIM_B = (2 * muX_B * muY_B + C1) * (2 * sigmaXY_B + C2) / ((Math.pow(muX_B, 2) + Math.pow(muY_B, 2) + C1) * (Math.pow(sigmaX_B, 2) + Math.pow(sigmaY_B, 2) + C2));
49
50      double SSIM_RGB = (SSIM_R + SSIM_G + SSIM_B) / 3.0;
51
52      return SSIM_RGB;
53  }

```

Gambar 4.2.8. Fungsi Metode *Structural Similarity Index*

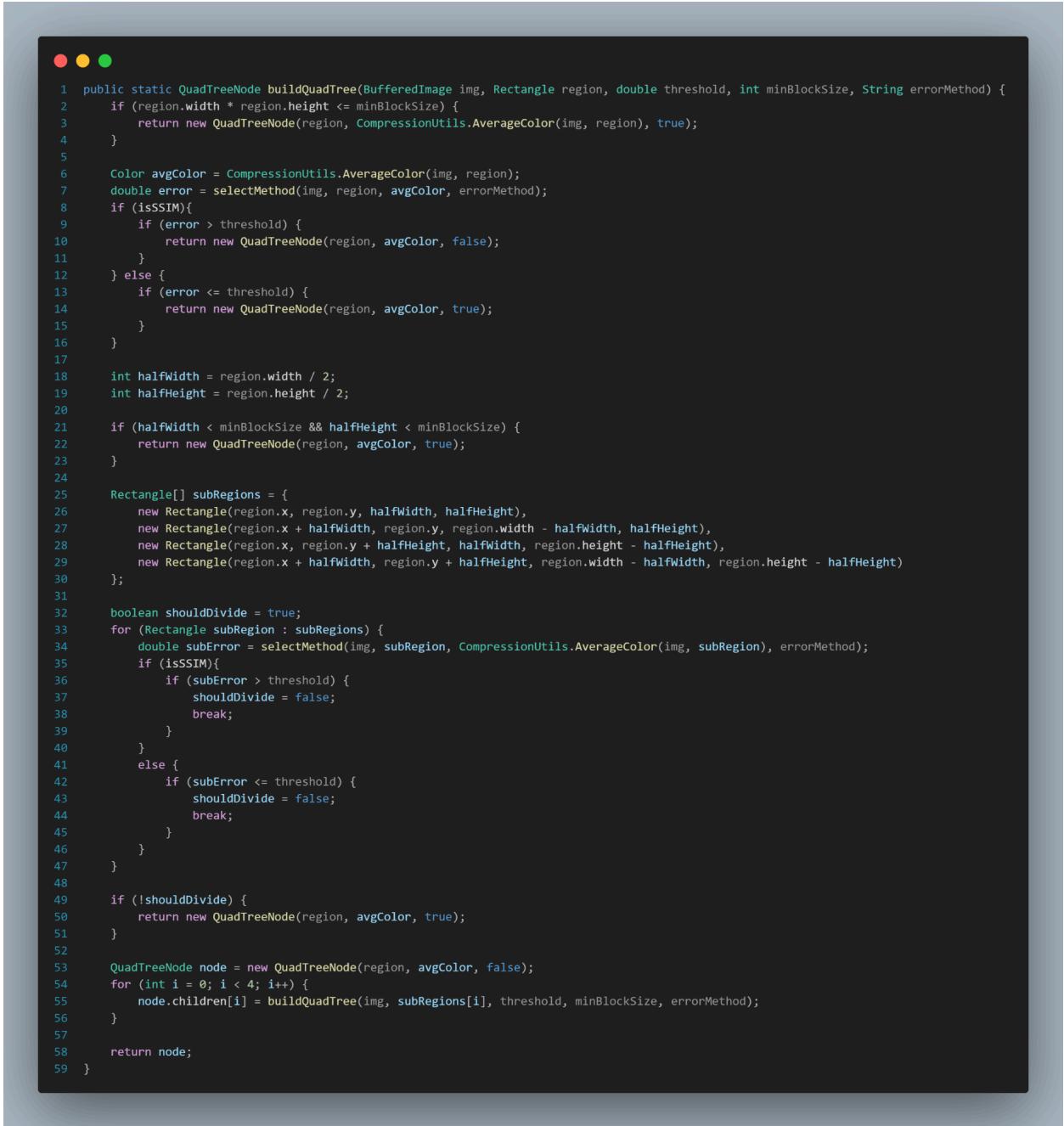
4.3. *Compression* (main)

Kelas utama yaitu *Compression* atau main memiliki suatu variabel global isSSIM yang dibuat untuk meng-*handle* metode SSIM karena SSIM memiliki penanganan kondisi yang berbeda dibanding opsi metode lainnya, sehingga struktur data kelas Compression terbentuk seperti dibawah ini:



Gambar 4.3.1. Struktur Data Kelas *Compression*

Terdapat fungsi utama yaitu fungsi *buildQuadTree* sebagai fungsi yang mengimplementasikan algoritma *divide and conquer* dalam memecah gambar menjadi beberapa wilayah yang dapat dikompresi secara individual berdasarkan kemiripannya selagi membentuk *QuadTreeNode* baru untuk menyimpan datanya. Fungsi ini diimplementasikan seperti:



```
1 public static QuadTreeNode buildQuadTree(BufferedImage img, Rectangle region, double threshold, int minBlockSize, String errorMethod) {
2     if (region.width * region.height <= minBlockSize) {
3         return new QuadTreeNode(region, CompressionUtils.AverageColor(img, region), true);
4     }
5
6     Color avgColor = CompressionUtils.AverageColor(img, region);
7     double error = selectMethod(img, region, avgColor, errorMethod);
8     if (isSSIM){
9         if (error > threshold) {
10             return new QuadTreeNode(region, avgColor, false);
11         }
12     } else {
13         if (error <= threshold) {
14             return new QuadTreeNode(region, avgColor, true);
15         }
16     }
17
18     int halfWidth = region.width / 2;
19     int halfHeight = region.height / 2;
20
21     if (halfWidth < minBlockSize && halfHeight < minBlockSize) {
22         return new QuadTreeNode(region, avgColor, true);
23     }
24
25     Rectangle[] subRegions = {
26         new Rectangle(region.x, region.y, halfWidth, halfHeight),
27         new Rectangle(region.x + halfWidth, region.y, region.width - halfWidth, halfHeight),
28         new Rectangle(region.x, region.y + halfHeight, halfWidth, region.height - halfHeight),
29         new Rectangle(region.x + halfWidth, region.y + halfHeight, region.width - halfWidth, region.height - halfHeight)
30     };
31
32     boolean shouldDivide = true;
33     for (Rectangle subRegion : subRegions) {
34         double subError = selectMethod(img, subRegion, CompressionUtils.AverageColor(img, subRegion), errorMethod);
35         if (isSSIM){
36             if (subError > threshold) {
37                 shouldDivide = false;
38                 break;
39             }
40         } else {
41             if (subError <= threshold) {
42                 shouldDivide = false;
43                 break;
44             }
45         }
46     }
47
48     if (!shouldDivide) {
49         return new QuadTreeNode(region, avgColor, true);
50     }
51
52     QuadTreeNode node = new QuadTreeNode(region, avgColor, false);
53     for (int i = 0; i < 4; i++) {
54         node.children[i] = buildQuadTree(img, subRegions[i], threshold, minBlockSize, errorMethod);
55     }
56
57     return node;
58 }
59 }
```

Gambar 4.3.2. Fungsi *buildQuadTree*

Lalu, juga terdapat fungsi pembantu utama yaitu fungsi *selectMethod* yang berguna untuk mempermudah pengguna selagi juga meng-*handle* input untuk pemilihan opsi metode pengukuran error yang disediakan, fungsi ini juga memiliki peran dalam mengubah variabel global *isSSIM* agar nanti dapat digunakan dalam memproses metode SSIM yang implementasinya seperti:



```
1 public static double selectMethod(BufferedImage img, Rectangle region, Color avgColor, String errorMethod) {  
2     switch (errorMethod) {  
3         case "1":  
4             return CompressionUtils.Variance(img, region, avgColor);  
5         case "2":  
6             return CompressionUtils.MAD(img, region, avgColor);  
7         case "3":  
8             return CompressionUtils.MaxPixelDifference(img, region, avgColor);  
9         case "4":  
10            return CompressionUtils.Entropy(img, region);  
11        case "5":  
12            isSSIM = true;  
13            return CompressionUtils.SSIM(img, region);  
14        default:  
15            throw new IllegalArgumentException("Metode error tidak valid.");  
16    }  
17}
```

Gambar 4.3.3. Fungsi *selectMethod*

Dengan demikian, main program dibuat dengan memanfaatkan segala fungsi yang telah dibuat sebelumnya dengan tampilan yang dibuat sesuai dengan ketentuan dalam spesifikasi yaitu dengan rincian kode:

```
1 public static void main(String[] args) {
2     Scanner scanner = new Scanner(System.in);
3
4     System.out.print("Masukkan alamat absolut gambar yang akan dikompresi: ");
5     String inputPath = scanner.nextLine();
6
7     String errorMethod = "";
8     while (true) {
9         System.out.println("\nPilih metode pengukuran error:");
10        System.out.println("1. Variance");
11        System.out.println("2. Mean Absolute Deviation (MAD)");
12        System.out.println("3. Max Pixel Difference");
13        System.out.println("4. Entropy");
14        System.out.println("5. Structural Similarity Index (SSIM)");
15        System.out.print("Pilih antara 1 - 5: ");
16        errorMethod = scanner.next();
17
18        if (errorMethod.equals("1") || errorMethod.equals("2") || errorMethod.equals("3") || errorMethod.equals("4") || errorMethod.equals("5")) {
19            break;
20        } else {
21            System.out.println("Masukan salah! Silakan pilih antara 1 sampai 5.");
22        }
23    }
24
25    File inputFile = new File(inputPath);
26    if (!inputFile.exists()) {
27        System.out.println("File gambar tidak ditemukan.");
28        return;
29    }
30
31    System.out.print("Masukkan ambang batas error: ");
32    double threshold = scanner.nextDouble();
33
34    System.out.print("Masukkan ukuran blok minimum: ");
35    int minBlockSize = scanner.nextInt();
36
37    System.out.print("Masukkan nama file output (tanpa ekstensi .png): ");
38    String outputFileName = scanner.next();
39    String outputPath = "test/" + outputFileName + ".png";
40
41    scanner.close();
42
43    try {
44        BufferedImage img = ImageIO.read(new File(inputPath));
45        long originalSize = inputFile.length();
46
47        long startTime = System.nanoTime();
48
49        QuadTreeNode root = buildQuadTree(img, new Rectangle(0, 0, img.getWidth(), img.getHeight()), threshold, minBlockSize, errorMethod);
50
51        BufferedImage compressedImg = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_RGB);
52        Graphics g = compressedImg.getGraphics();
53        CompressionUtils.drawCompressedImage(g, root);
54        g.dispose();
55        ImageIO.write(compressedImg, "png", new File(outputPath));
56
57        long endTime = System.nanoTime();
58        long executionTime = (endTime - startTime) / 1000000;
59
60        long compressedSize = new File(outputPath).length();
61        double compressionRatio = (1.0 - ((double) compressedSize / originalSize)) * 100;
62        int depth = QuadTreeNode.countDepths(root);
63        int nodeCount = QuadTreeNode.countNodes(root);
64
65        System.out.println("\n== HASIL KOMPRESI ==");
66        System.out.println("Waktu eksekusi: " + executionTime + " ms");
67        System.out.println("Ukuran gambar sebelum: " + originalSize / 1024 + " KB");
68        System.out.println("Ukuran gambar setelah: " + compressedSize / 1024 + " KB");
69        System.out.printf("Persentase kompresi: %.2f%%\n", compressionRatio);
70        System.out.println("Kedalaman pohon: " + depth);
71        System.out.println("Banyak simpul pada pohon: " + nodeCount);
72        System.out.println("Gambar hasil kompresi tersimpan di: " + outputPath);
73
74    } catch (Exception e) {
75        System.out.println("Terjadi kesalahan: " + e.getMessage());
76    }
77 }
```

Gambar 4.3.4. Main Program

BAB V

EKSPERIMEN & ANALISIS

5.1 Eksperimen

1. Metode varians, 20, 5

Sebelum	Setelah
	
<p>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test1.jpg Pilih metode pengukuran error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Pilih antara 1 - 5: 1 Masukkan ambang batas error: 200 Masukkan ukuran blok minimum: 8 Masukkan nama file output (tanpa ekstensi .png): output1 ==== HASIL KOMPRESI ==== Waktu eksekusi: 1252 ms Ukuran gambar sebelum: 1378 KB Ukuran gambar setelah: 126 KB Persentase kompresi: 90,82% Kedalaman pohon: 7 Banyak simpul pada pohon: 4361 Gambar hasil kompresi tersimpan di: test/output1.png</p>	

2. Metode MAD, 10, 6

Sebelum	Setelah
	

```

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test2.jpg
Pilih metode pengukuran error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
Pilih antara 1 - 5: 2
Masukkan ambang batas error: 10
Masukkan ukuran blok minimum: 6
Masukkan nama file output (tanpa ekstensi .png): output2

==== HASIL KOMPRESI ====
Waktu eksekusi: 1506 ms
Ukuran gambar sebelum: 235 KB
Ukuran gambar setelah: 164 KB
Persentase kompresi: 30,19%
Kedalaman pohon: 8
Banyak simpul pada pohon: 21373
Gambar hasil kompresi tersimpan di: test/output2.png

```

3. Metode Max Pixel Difference, 10, 4

Sebelum	Setelah
	

```

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test3.jpg
Pilih metode pengukuran error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
Pilih antara 1 - 5: 3
Masukkan ambang batas error: 10
Masukkan ukuran blok minimum: 4
Masukkan nama file output (tanpa ekstensi .png): output3

==== HASIL KOMPRESI ====
Waktu eksekusi: 2154 ms
Ukuran gambar sebelum: 508 KB
Ukuran gambar setelah: 298 KB
Persentase kompresi: 41,34%
Kedalaman pohon: 9
Banyak simpul pada pohon: 61385
Gambar hasil kompresi tersimpan di: test/output3.png

```

4. Metode Entropy, 0.1, 10

Sebelum	Setelah
	

Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test4.jpg

Pilih metode pengukuran error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Pilih antara 1 - 5: 4

Masukkan ambang batas error: 0,1

Masukkan ukuran blok minimum: 10

Masukkan nama file output (tanpa ekstensi .png): output4

==== HASIL KOMPRESI ====
Waktu eksekusi: 1879 ms
Ukuran gambar sebelum: 581 KB
Ukuran gambar setelah: 344 KB
Persentase kompresi: 40,78%
Kedalaman pohon: 7
Banyak simpul pada pohon: 21097
Gambar hasil kompresi tersimpan di: test/output4.png

5. Metode SSIM, 0.3, 6

Sebelum	Setelah
	

```
Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test5.jpg
```

Pilih metode pengukuran error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)

Pilih antara 1 - 5: 5

Masukkan ambang batas error: 0,3

Masukkan ukuran blok minimum: 6

Masukkan nama file output (tanpa ekstensi .png): output5

```
==== HASIL KOMPRESI ===
```

Waktu eksekusi: 2986 ms

Ukuran gambar sebelum: 942 KB

Ukuran gambar setelah: 274 KB

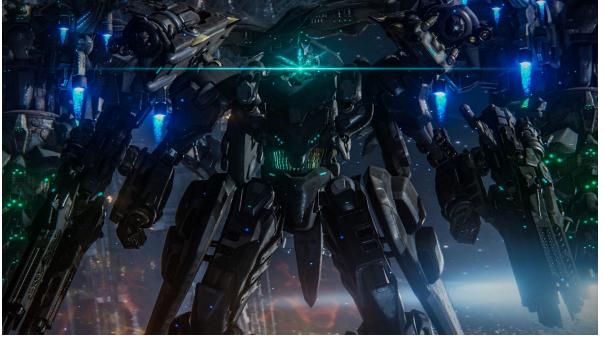
Persentase kompresi: 70,82%

Kedalaman pohon: 8

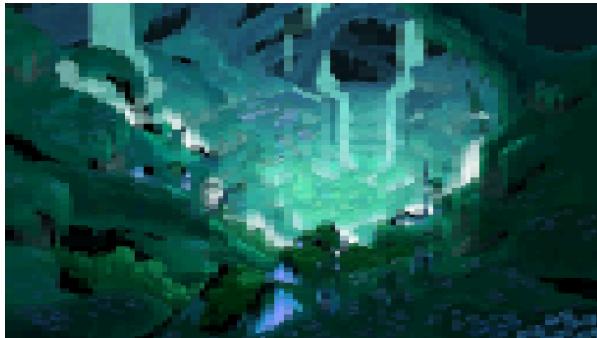
Banyak simpul pada pohon: 87309

Gambar hasil kompresi tersimpan di: test/output5.png

6. Metode Max Pixel Difference, 20, 6

Sebelum	Setelah
	
<pre>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test6.jpeg Pilih metode pengukuran error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Pilih antara 1 - 5: 3 Masukkan ambang batas error: 20 Masukkan ukuran blok minimum: 6 Masukkan nama file output (tanpa ekstensi .png): output6 ==== HASIL KOMPRESI ==== Waktu eksekusi: 2314 ms Ukuran gambar sebelum: 293 KB Ukuran gambar setelah: 190 KB Persentase kompresi: 35,20% Kedalaman pohon: 8 Banyak simpul pada pohon: 17953 Gambar hasil kompresi tersimpan di: test/output6.png</pre>	

7. Metode Entropy, 0,7, 7

Sebelum	Setelah
	
<pre>Masukkan alamat absolut gambar yang akan dikompresi: C:\Users\IRWAN\Pictures\Warframe\test7.jpg Pilih metode pengukuran error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Pilih antara 1 - 5: 4 Masukkan ambang batas error: 0,9 Masukkan ukuran blok minimum: 6 Masukkan nama file output (tanpa ekstensi .png): output7 ==== HASIL KOMPRESI ==== Waktu eksekusi: 677 ms Ukuran gambar sebelum: 65 KB Ukuran gambar setelah: 45 KB Percentase kompresi: 30,73% Kedalaman pohon: 7 Banyak simpul pada pohon: 12493 Gambar hasil kompresi tersimpan di: test/output7.png</pre>	

5.2 Analisis

Kompresi gambar mencakup beberapa proses. Program mengandalkan struktur QuadTree berbasis algoritma *divide and conquer*. Proses QuadTree dimulai dari perhitungan warna rata-rata yang digunakan dalam perhitungan galat / error. Perhitungan error dilakukan dengan salah satu dari berbagai metode IQA. Galat yang telah dihitung akan dibandingkan dengan ambang batas error yang diinput pengguna. Bila galat masih diatas ambang batas error dan ukuran blok masih diatas ukuran blok minimum, blok akan dibagi menjadi 4 sub-blok.

Agar bisa memperoleh kompleksitas algoritma dari program yang telah dibuat, diperlukan *breakdown* dari masing-masing fungsi. Setiap metode IQA memiliki kompleksitas

waktu $O(n^2)$ karena penggunaan for loop tingkat 2 di setiap proses perhitungannya. Perhitungan memanfaatkan dua titik koordinat (x dan y) yang diakses dari loop yang bertingkat.

Dalam proses pembuatan QuadTree, gambar semula diubah menjadi 4 blok yang berukuran sama. Lalu setiap blok tersebut akan terbagi lagi menjadi 4 blok. Proses ini berlangsung hingga batas ukuran blok minimum atau sudah memiliki warna yang cocok. Proses mengandalkan beberapa loop ($O(n)$) untuk menentukan bila blok yang telah terbagi harus dibagi lagi atau tidak. Untuk melanjutkan proses pembagian blok, digunakan juga proses rekursi.

Berdasarkan master theorem untuk rekursi, kompleksitas waktu dapat dinyatakan sebagai berikut.

$$T(n) = aT(n/b) + f(n)$$

Dimana,

a : jumlah pembagian submasalah

n/b : ukuran dari setiap submasalah

$f(n)$: kompleksitas pekerjaan diluar rekursi

Karena setiap rekursi membagi sebuah blok menjadi 4 blok dan proses QuadTree melibatkan blok yang berukuran $W \times H$, maka dapat diketahui bahwa kompleksitas waktu dari QuadTree adalah sebagai berikut.

$$T(n) = 4T(n^2/4) + O(n^2)$$

LAMPIRAN

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
2	Program berhasil dijalankan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	<input checked="" type="checkbox"/>	<input type="checkbox"/>
4	Mengimplementasi seluruh metode perhitungan error wajib	<input checked="" type="checkbox"/>	<input type="checkbox"/>
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan	<input type="checkbox"/>	<input checked="" type="checkbox"/>
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	<input checked="" type="checkbox"/>	<input type="checkbox"/>
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	<input type="checkbox"/>	<input checked="" type="checkbox"/>
8	Program dan laporan dibuat (kelompok) sendiri	<input checked="" type="checkbox"/>	<input type="checkbox"/>

Github Repository: [TKurr/Tucil2_13523154_13523163](https://github.com/TKurr/Tucil2_13523154_13523163)

DAFTAR PUSTAKA

Tarjan, R. E., dan M. H. Goldwasser. "Analysis of divide-and-conquer algorithms using recurrences," *IEEE Transactions on Education*, vol. 47, no. 2, pp. 134–142, April 2004, doi: 10.1109/TE.2004.1273769.

GeeksforGeeks. "Advanced master theorem for divide and conquer recurrences," url: <https://www.geeksforgeeks.org/advanced-master-theorem-for-divide-and-conquer-recurrences>

Roughgarden, Tim. "CS161: Design and Analysis of Algorithms – Lecture 3: Recurrences," Stanford University. url: <https://web.stanford.edu/class/archive/cs/cs161/cs161.1168/lecture3.pdf>

GeeksforGeeks. "Quad Tree," url: <https://www.geeksforgeeks.org/quad-tree/>

Rinaldi Munir. "Tugas Kecil II – Struktur Data dan Algoritma: Kompresi Citra dengan Quad Tree," Institut Teknologi Bandung. (2025). url: <https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tucil2-Stima-2025.pdf>