

IF2211 Strategi Algoritma
Laporan Tugas Kecil 2



Disusun oleh:
Rafael Marchel Darma Wijaya (13523146)

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

Daftar Isi

Daftar Isi	2
BAB I	
DESKRIPSI MASALAH	4
BAB II	
DASAR TEORI	8
2.1. Divide and Conquer	8
2.2. Kompresi Gambar	9
2.3. Struktur Data Quadtree	10
2.4. Image Quality Assessment (IQA)	11
2.4.1. Variance (Varian)	11
2.4.2. Mean Absolute Deviation (MAD)	12
2.4.3. Max Pixel Difference (MPD)	14
2.4.4. Entropy	15
2.4.5. Structural Similarity Index (SSIM)	16
BAB III	
IMPLEMENTASI PROGRAM	18
3.1. Repotori Program	18
3.2. Implementasi Algoritma	18
3.3. Kode Sumber Program	21
3.3.1. InputManager	21
3.3.2. Quadtree	29
3.3.1. Main	39
BAB IV	
EKSPERIMENT DAN ANALISIS	46
4.1. Pengujian Program	46
4.2. Analisis Algoritma	63
BAB V	
LAMPIRAN	66
Pustaka	67

BAB I

DESKRIPSI MASALAH

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.

Masalah dalam tugas ini adalah kompresi gambar menggunakan *Quadtree* sebagai bentuk penyelesaian masalah dengan metode *divide and conquer*. Implementasi ini penulis buat dalam bahasa pemrograman Java (CLI) dengan beberapa parameter sebagai berikut:

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error.
3. [INPUT] ambang batas
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi.
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [OUTPUT] waktu eksekusi.
8. [OUTPUT] ukuran gambar sebelum.
9. [OUTPUT] ukuran gambar setelah.
10. [OUTPUT] persentase kompresi.
11. [OUTPUT] kedalaman pohon.
12. [OUTPUT] banyak simpul pada pohon.
13. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.

Prosedur yang digunakan dalam implementasi algoritma ini adalah sebagai berikut:

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.

- 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.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan *threshold*:

- Jika variansi di atas *threshold*, 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.
- 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:

- Error blok berada di bawah *threshold*.

- 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.

BAB II

DASAR TEORI

2.1. Divide and Conquer

Divide and Conquer adalah salah satu strategi algoritma yang sangat penting dalam dunia pemrograman, terutama dalam pemrosesan data yang bersifat rekursif atau memiliki struktur hierarkis. Strategi ini memecah suatu permasalahan besar menjadi beberapa sub-masalah yang lebih kecil, menyelesaikan setiap sub-masalah tersebut secara terpisah (seringkali secara rekursif), dan kemudian menggabungkan hasil-hasilnya untuk membentuk solusi dari permasalahan awal.

Proses *Divide and Conquer* terdiri dari tiga tahap utama:

1. *Divide*: Masalah dipecah menjadi beberapa sub-masalah yang lebih kecil namun serupa dengan masalah aslinya.
2. *Conquer*: Setiap sub-masalah diselesaikan, biasanya secara rekursif. Jika sub-masalah cukup kecil, maka diselesaikan secara langsung (*base case*).
3. *Combine*: Hasil dari sub-masalah digabungkan untuk membentuk solusi akhir dari masalah asli.

Dalam konteks pemrosesan gambar, pendekatan *Divide and Conquer* sangat ideal karena gambar merupakan representasi data dua dimensi yang dapat direpresentasikan sebagai matriks piksel. Dengan membagi gambar menjadi blok-blok kecil dan menangani setiap blok secara independen, kita dapat menerapkan logika rekursif untuk menyederhanakan atau mengkompresi gambar berdasarkan karakteristik lokal dari tiap blok. Strategi ini sejalan dengan konsep bagian-bagian yang memiliki nilai warna yang seragam dapat diwakili secara sederhana tanpa mengorbankan kualitas visual.

Keuntungan dari pendekatan ini adalah efisiensi dalam memproses data besar secara lokal, skalabilitas algoritma, serta kemudahan implementasi dalam struktur data berbasis pohon seperti *Quadtree*. Proses rekursif yang berhenti berdasarkan kondisi tertentu (seperti kesamaan blok atau ukuran minimum) juga menjadikan *Divide and Conquer* sangat fleksibel dalam menyeimbangkan antara akurasi dan efisiensi komputasi.

2.2. Kompresi Gambar

Kompresi gambar adalah proses mengurangi ukuran *file* gambar digital tanpa secara signifikan menurunkan kualitas visualnya. Tujuan utama dari kompresi adalah efisiensi penyimpanan. Kompresi dapat diklasifikasikan menjadi dua kategori utama:

1. Kompresi *Lossless*, yang mempertahankan semua informasi asli dan memungkinkan rekonstruksi gambar.
2. Kompresi *Lossy*, yang menghilangkan sebagian informasi untuk mendapatkan efisiensi yang lebih tinggi, dengan kualitas yang lebih rendah.

Pada gambar digital, piksel disimpan dalam bentuk nilai intensitas atau warna dalam model seperti RGB. Kompresi dilakukan dengan cara mengevaluasi pola dan redundansi dari data ini. Dalam kompresi berbasis area, seperti yang dilakukan *Quadtree*, gambar dibagi menjadi blok-blok dan setiap blok diuji keragamannya. Jika blok dianggap cukup seragam (berdasarkan nilai error), maka blok tersebut tidak perlu disimpan piksel demi piksel, melainkan dapat direpresentasikan dengan nilai rata-ratanya saja.

Kompresi gambar dilakukan dengan membagi gambar berdasarkan kuadran hingga tiap bagian cukup homogen. Proses ini menghasilkan struktur data pohon, yang mencerminkan hierarki dalam

gambar, memungkinkan penghilangan data yang tidak signifikan dan penghematan ruang.

2.3. Struktur Data Quadtree

Quadtree adalah struktur data pohon (tree) yang digunakan untuk membagi ruang dua dimensi menjadi empat bagian (kuadran). Pada setiap tingkat pohon, setiap node dapat memiliki empat anak yang mewakili subdivisi dari area node tersebut. Struktur quadtree sangat cocok digunakan dalam aplikasi pengolahan citra dan grafika komputer, terutama dalam representasi spasial dan kompresi gambar.

Pada pengolahan gambar, quadtree digunakan untuk merepresentasikan gambar dalam bentuk blok-blok persegi yang bersifat homogen (memiliki intensitas warna atau struktur yang mirip). Prosesnya adalah sebagai berikut:

1. Cek Homogenitas: Periksa apakah suatu blok gambar homogen, yaitu apakah seluruh piksel dalam blok memiliki warna atau intensitas yang sama atau mirip.
2. Bagi Jika Tidak Homogen: Jika blok tidak homogen, bagi blok menjadi empat bagian kuadran.
3. Rekursif: Lakukan pemeriksaan dan pembagian ini secara rekursif sampai blok-blok yang diperoleh dianggap cukup homogen atau mencapai ukuran terkecil yang diizinkan.
4. Simpan sebagai Pohon: Struktur yang dihasilkan kemudian direpresentasikan sebagai pohon di mana setiap node menyimpan informasi blok, dan daun-daun (leaf nodes) merepresentasikan blok-blok homogen akhir.

Quadtree bersifat adaptif terhadap gambar. Wilayah yang kompleks dibagi lebih banyak, sedangkan wilayah homogen tidak perlu banyak dibagi.

2.4. Image Quality Assessment (IQA)

Image Quality Assessment adalah proses untuk mengevaluasi seberapa baik kualitas visual dari suatu gambar. Dalam konteks kompresi gambar, IQA digunakan untuk membandingkan gambar asli (referensi) dengan gambar hasil proses, dan mengukur sejauh mana kualitas gambar tersebut berubah atau menurun.

IQA terbagi menjadi dua kategori utama:

1. *Subjective IQA*: Melibatkan manusia untuk menilai kualitas secara visual.
2. *Objective IQA*: Menggunakan rumus matematis untuk menghitung selisih antara gambar asli dan hasil.

2.4.1. Variance (Varian)

Variance adalah ukuran penyebaran nilai piksel dari rata-ratanya. Dalam konteks IQA, *variance* bisa digunakan untuk memahami konsistensi nilai kesalahan (error) antara dua gambar.

Untuk menghitung nilai variance untuk setiap kanal warna dalam satu blok warna, digunakan rumus:

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

σ_c^2 = Nilai varians kanal warna c

$P_{i,c}$ = Nilai kanal warna c pada piksel ke-i

μ_c = Nilai rata-rata untuk kanal warna c

N = Banyak piksel

Nilai *variance* RGB dihitung untuk setiap kanal warna (R, G, B) kemudian dicari rata-ratanya.

$$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

Untuk mencari batas minimum dan maksimum dari metode penghitungan error dengan *variance*, perlu dicari nilainya ketika satu area gambar memiliki warna yang sama dan ketika warna paling bervariasi. Ketika seluruh warna homogen, maka kuadrat selisih akan bernilai 0 (semua piksel bernilai sama, selisihnya 0). Ketika setengah piksel bernilai 0 (minimum) dan setengahnya lagi bernilai 255 (maksimum), rata-ratanya adalah 127,5 dan selisihnya juga 127,5. Maka, kuadrat selisih bernilai $127^2 = 16.256,25$.

2.4.2. Mean Absolute Deviation (MAD)

MAD mengukur rata-rata dari nilai mutlak deviasi piksel terhadap gambar. Ini adalah versi sederhana dari metode *variance*, yang membedakan hanyalah selisih nilai dimutlakkan

bukan dikuadratkan. Namun, metode ini kurang sensitif dalam mendeteksi *outlier*.

Untuk menghitung nilai MAD untuk setiap kanal warna dalam satu blok warna, digunakan rumus:

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

MAD_c = Nilai MAD untuk setiap kanal warna c

$P_{i,c}$ = Nilai piksel kanal warna c pada piksel ke-i

μ_c = Nilai rata-rata untuk kanal warna c

N = Banyak piksel

Nilai MAD RGB dihitung untuk setiap kanal warna (R, G, B) kemudian dicari rata-ratanya.

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

Untuk mencari batas minimum dan maksimum dari metode penghitungan error dengan MAD, caranya hampir sama hanya saja tidak dikuadratkan. Ketika seluruh warna homogen, maka minimum mutlak selisih akan bernilai 0. Ketika setengah piksel bernilai 0 (minimum) dan setengahnya lagi bernilai 255

(maksimum), rata-ratanya adalah 127,5 sehingga maksimum mutlak selisihnya adalah 127,5.

2.4.3. Max Pixel Difference (MPD)

MPD adalah perbedaan antara piksel maksimum dan minimum pada gambar. Ini memberikan informasi tentang seberapa kontras nilai piksel pada sampel gambar. Namun, metode ini menjadi tidak akurat jika ada *noise*, karena sensitif dengan *outlier*.

Untuk menghitung nilai MPD untuk setiap kanal warna dalam satu blok warna, digunakan rumus:

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

D_c = Nilai selisih maksimum dan minimum untuk kanal warna c

$P_{i,c}$ = Nilai piksel pada kanal warna c

Nilai MPD RGB dihitung untuk setiap kanal warna (R, G, B) kemudian dicari rata-ratanya.

$$D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

Nilai minimum MPD adalah ketika semua piksel bernilai sama (nilai maksimum dan minimum piksel sama) sehingga selisihnya adalah 0. Nilai maksimum MPD adalah ketika terdapat piksel bernilai 0 dan piksel bernilai 255 sehingga selisihnya adalah 255.

2.4.4. Entropy

Entropy adalah metode untuk mengukur ketidakteraturan warna pada gambar. Metode ini memanfaatkan informasi jumlah kemunculan warna untuk setiap warna dalam gambar dan membentuk histogram untuk memetakan probabilitas kemunculan warna. Metode ini bisa saja tidak akurat jika gambar memiliki banyak *noise*.

Untuk menghitung nilai *entropy* untuk setiap kanal warna dalam satu blok warna, digunakan rumus:

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

H_c = Nilai entropi untuk kanal warna c

$P_c(i)$ = Probabilitas dengan nilai i pada histogram gambar untuk kanal warna c

Nilai entropi RGB dihitung untuk setiap kanal warna (R, G, B) kemudian dicari rata-ratanya.

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

Ketika semua nilai piksel bernilai sama, maka histogram gambar akan terkonsentrasi pada satu nilai saja sehingga nilai probabilitasnya 1. Maka, nilai minimum entropi adalah,

$$-1 \log_2(1) = 0$$

Ketika semua nilai piksel hanya muncul sekali, maka histogram gambar akan tersebar pada setiap nilai sehingga nilai probabilitasnya $1/N$. Maka, nilai maksimum entropi adalah,

$$-256 \frac{1}{256} \log_2\left(\frac{1}{256}\right) = 8, \text{ untuk } N = 256 [0, 255]$$

2.4.5. Structural Similarity Index (SSIM)

SSIM adalah metode yang menggunakan gambar lain sebagai referensi dan pembanding dalam perhitungan error. Semakin dekat suatu gambar dengan gambar referensi, semakin tinggi nilai SSIM. Metode SSIM ini membandingkan gambar berdasarkan fungsi komparasi luminansi (*luminance*), fungsi komparasi kontras, dan fungsi komparasi struktur. Berdasarkan ketiga fungsi komparasi tersebut, fungsi SSIM adalah sebagai berikut:

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

$\mu_{x,c}$ = Rata-rata sinyal gambar x di kanal warna c

$\mu_{y,c}$ = Rata-rata sinyal gambar y di kanal warna c

$\sigma_{x,c}^2$ = Varians sinyal gambar x di kanal warna c

$\sigma_{y,c}^2$ = Varians sinyal gambar y di kanal warna c

$\sigma_{xy,c}$ = Kovarians sinyal gambar x dan y di kanal warna c

C_1 = Konstanta perbandingan luminansi

C_2 = Konstanta perbandingan kontras

Nilai SSIM RGB dihitung untuk setiap kanal warna (R, G, B) kemudian dicari dengan pembobotan BT.601-7 / Rec.601-7.

$$SSIM_{RGB} = 0.299 \cdot SSIM_R + 0.587 \cdot SSIM_G + 0.114 \cdot SSIM_B$$

Metode SSIM ini memiliki rentang nilai $[-1, 1]$. Namun, pada konteks kompresi gambar quadtree, rumus SSIM telah disederhanakan karena gambar yang dikompresi memiliki warna konstan untuk satu blok. Sehingga, varians gambar kompresi dan kovarians bernilai 0 dan nilai rata-rata sinyal gambar akan sama. Metode SSIM yang telah disederhanakan ini akan memiliki rentang $[0, 1]$.

BAB III

IMPLEMENTASI PROGRAM

3.1. Repotori Program

Repository Program dapat diakses melalui tautan Github berikut:
https://github.com/V-Kleio/Tucil2_13523146

3.2. Implementasi Algoritma

Secara garis program, algoritma yang penulis buat terbagi menjadi tiga *file* kode. Pertama adalah kode untuk mengatur masukkan dari pengguna. Kedua adalah kode untuk pembuatan *quadtree* dan kompresi. Ketiga adalah kode *main* sebagai program utama dan *entry point*, serta menggunakan kode pertama dan kedua untuk garis besar eksekusi program.

Program pertama-tama akan melakukan manajemen masukkan dari pengguna. Terdapat objek yang bernama InputManager untuk mengatur hal ini. InputManager berisi atribut-atribut untuk menyimpan masukkan user dan method-method untuk meminta masukkan tersebut. Fungsi-fungsi ini dilengkapi dengan validasi input agar program tetap berjalan walaupun input tidak valid dan memastikan input agar tidak merusak program. Method dalam InputManager juga berperan sebagai antarmuka dengan pengguna. Objek dari InputManager diinstansiasi di *main* program.

Setelah mendapatkan seluruh parameter masukkan, main program akan menginstansiasi objek quadtree untuk melakukan kompresi. Di dalam class Quadtree, terdapat private class Node yang melambangkan tiap simpul pada pohon. Tiap node menyimpan informasi antara lain: koordinat x dan y pada gambar asli yang merupakan posisi starting piksel dalam node (piksel pojok kiri atas), width dan height yang merupakan lebar dan tinggi sub-gambar pada node, empat node anak (top left, top right, bottom left, bottom right)

jika node dibagi lagi menjadi empat kuadran, dan status isLeaf untuk menentukan apakah node tersebut memiliki anak atau tidak.

Inti utama dari algoritma quadtree ini adalah fungsi buildTree() yang membuat pohon representasi kompresi gambar secara rekursif. Fungsi ini awalnya akan membuat satu node berdasarkan masukkan parameter x, y, width, dan height. Lalu fungsi ini akan melakukan kalkulasi error sesuai dengan metode yang telah dipilih pengguna. Jika error kurang dari error threshold (masukkan pengguna) atau luas node kurang dari luas blok minimum atau nilai width/height kurang dari sama dengan 1, status node akan menjadi leaf dan tidak akan dibagi 4. Lalu node akan menghitung average color yang akan menjadi warna node tersebut. Namun jika tidak demikian, node akan dibagi menjadi 4 node lagi dengan width dan height dibagi 2, posisi (x, y) untuk node kiri atas, (x + width/2, y) untuk node kanan atas, (x, y + height/2) untuk node kiri bawah, dan (x + width/2, y + height/2) untuk node kanan bawah. Pembagian node ini juga menggunakan fungsi buildTree (rekursif).

Perhitungan error untuk setiap region node dilakukan dengan metode yang telah dipilih oleh pengguna dan dibandingkan dengan error threshold. Hal ini membuat area gambar yang memiliki karakteristik homogen, cukup direpresentasikan sebagai satu nilai rata-rata (nilai node). Jika sudah di bawah threshold, node akan menghitung rata-rata RGB value pada region node tersebut. Setelah rekursi berakhir, pohon akan digambar ulang.

Menggambar pohon dilakukan juga secara rekursif dengan mengunjungi setiap node. Namun, tidak semua node digambar. Hanya node yang merupakan leaf node yang akan digambar. Fungsi gambar akan menggambar area node satu persatu dengan nilai warna rata-rata. Hasil gambar ini merupakan hasil kompresi dari algoritma quadtree.

Hasil gambar akan disimpan pada file dengan ekstensi yang sama dengan gambar input dan berada di alamat yang dimasukkan pengguna. Main program juga melakukan kalkulasi dan mendapatkan statistik lalu menampilkannya setelah gambar berhasil dikompresi.

[Bonus] Main program juga akan mengurus algoritma untuk target kompresi. Jika pengguna menggunakan fitur ini, main program akan mencoba untuk mencapai target kompresi dengan cara naif yaitu *trial and error*. Pengguna awalnya akan diminta memasukkan banyaknya percobaan yang akan dilakukan. Kemudian, main akan mencoba melakukan kompresi dengan threshold awal berdasarkan metode (2% dari rentang minimum dan maksimum threshold). Setelah selesai, gambar hasil kompresi akan disimpan secara in-memory dan tidak di-write ke disk menggunakan ByteArrayOutputStream. Hal ini dilakukan agar dapat menghitung size gambar tanpa harus menyimpan gambar tersebut ke disk. Nilai threshold akan dinaikkan atau diturunkan berdasarkan jarak persentase kompresi saat ini dan persentase target. Program akan mengulangi hal ini sebanyak nilai banyaknya percobaan yang dimasukkan pengguna. Kemudian, program akan menggunakan error threshold yang akan menghasilkan persentase kompresi yang di atas (kalau bisa selama percobaan) dan paling dekat dengan target kompresi.

3.3. Kode Sumber Program

3.3.1. InputManager

```
● ● ●  
1 public class InputManager {  
2     private BufferedImage image = null;  
3     private ErrorCalculationMethod method;  
4     private float errorThreshold;  
5     private int minimumBlockSize;  
6     private String imageOutputPath;  
7     private String imageInputPath;  
8     private double minCompressionPercentage;  
9     private int maxSearchAttempts = 10;
```

Atribut-atribut dari class InputManager.

```
1 public void getUserImage(Scanner userInput) {
2     System.out.println("Masukkan alamat absolut gambar.");
3     boolean valid = false;
4     while (!valid) {
5         System.out.print("> ");
6         String imagePath = userInput.nextLine();
7
8         File imageFile = new File(imagePath);
9         if (!imageFile.exists()) {
10             System.out.println("File tidak ditemukan, silakan masukkan alamat yang valid!");
11             continue;
12         }
13         if (!imageFile.canRead()) {
14             System.out.println("File tidak dapat dibaca, silakan masukkan alamat dengan akses read!");
15             continue;
16         }
17         try {
18             image = ImageIO.read(imageFile);
19             if (image == null) {
20                 System.out.println("Format gambar tidak didukung atau file rusak");
21             } else {
22                 imageInputPath = imagePath;
23                 valid = true;
24             }
25         } catch (IOException e) {
26             System.out.println("Terjadi kesalahan saat membaca gambar: " + e.getMessage());
27         }
28     }
29 }
```

Method untuk menampilkan interface yang meminta input alamat absolut gambar original. Hal-hal yang divalidasi adalah: keberadaan file, keterbacaan file (masalah Read permission), dan keberhasilan membaca gambar.

```
● ● ●
1 public void getUserErrorMethod(Scanner userInput) {
2     System.out.println("Pilih metode perhitungan error.");
3     System.out.println("1. Metode variance");
4     System.out.println("2. Metode mean absolute deviation (MAD)");
5     System.out.println("3. Metode max pixel difference");
6     System.out.println("4. Metode entropy");
7     System.out.println("5. Metode SSIM");
8
9     int input = 0;
10    boolean valid = false;
11    while (!valid) {
12        System.out.print("(1 - 5)> ");
13        String line = userInput.nextLine();
14        try {
15            input = Integer.parseInt(line);
16            if (input ≥ 1 && input ≤ 5) {
17                valid = true;
18            } else {
19                System.out.println("Input harus berupa angka 1 sampai 5.");
20            }
21        } catch (NumberFormatException e) {
22            System.out.println("Input tidak valid, silakan masukkan integer.");
23        }
24    }
25
26    switch (input) {
27        case 1 → method = ErrorCalculationMethod.VARIANCE;
28        case 2 → method = ErrorCalculationMethod.MAD;
29        case 3 → method = ErrorCalculationMethod.MAX_PIXEL_DIFFERENCE;
30        case 4 → method = ErrorCalculationMethod.ENTROPY;
31        case 5 → method = ErrorCalculationMethod.SSIM;
32    }
33 }
```

Method untuk menampilkan interface yang meminta input metode penghitungan error. Hal-hal yang divalidasi adalah: range integer (1 - 5) dan tipe data masukkan.

```
● ● ●
1 public void getUserMinimumCompressionPercentage(Scanner userInput) {
2     System.out.println("Masukkan persentase kompresi minimum yang diinginkan (0.0 - 1.0).");
3     System.out.println("0.0 = fitur ini dinonaktifkan, masukkan parameter secara manual");
4     System.out.print("> ");
5
6     boolean valid = false;
7     while (!valid) {
8         String input = userInput.nextLine();
9         try {
10             double value = Double.parseDouble(input);
11             if (value < 0.0 || value > 1.0) {
12                 System.out.println("Nilai harus antara 0.0 dan 1.0");
13                 System.out.print("> ");
14                 continue;
15             }
16
17             minCompressionPercentage = value;
18             valid = true;
19         } catch (NumberFormatException e) {
20             System.out.println("Input tidak valid, masukkan angka desimal.");
21             System.out.print("> ");
22         }
23     }
24 }
```

Method untuk menampilkan interface yang meminta input target kompresi. Hal-hal yang divalidasi adalah: range double (0.0 – 1.0) dan tipe data masukkan.

```
1 public void getUserMaxSearchAttempts(Scanner userInput) {
2     System.out.println("Masukkan jumlah percobaan maksimum untuk pencarian persentase kompresi.");
3     System.out.println("Default: 10");
4     System.out.print("> ");
5
6     boolean valid = false;
7     while (!valid) {
8         String input = userInput.nextLine();
9
10        if (input.trim().isEmpty()) {
11            System.out.println("Menggunakan jumlah percobaan default: 10");
12            maxSearchAttempts = 10;
13            valid = true;
14            continue;
15        }
16
17        try {
18            int value = Integer.parseInt(input);
19            if (value < 1) {
20                System.out.println("Jumlah percobaan minimal 1.");
21                System.out.print("> ");
22            } else {
23                maxSearchAttempts = value;
24                valid = true;
25            }
26        } catch (NumberFormatException e) {
27            System.out.println("Input tidak valid, masukkan angka bulat.");
28            System.out.print("> ");
29        }
30    }
31 }
```

Method untuk menampilkan interface yang meminta banyaknya percobaan untuk mencapai target kompresi. Hal-hal yang divalidasi adalah: range integer (>0) dan tipe data masukkan. Jika pengguna tidak memasukkan input, menggunakan jumlah default (10).

```
1 public void getUserThreshold(Scanner userInput) {
2     float minThreshold = 0.0f, maxThreshold = 0.0f;
3
4     switch (method) {
5         case ErrorCalculationMethod.VARIANCE → {
6             minThreshold = 0.0f;
7             maxThreshold = 16256.25f;
8         }
9         case ErrorCalculationMethod.MAD → {
10            minThreshold = 0.0f;
11            maxThreshold = 127.5f;
12        }
13        case ErrorCalculationMethod.MAX_PIXEL_DIFFERENCE → {
14            minThreshold = 0.0f;
15            maxThreshold = 255;
16        }
17        case ErrorCalculationMethod.ENTROPY → {
18            minThreshold = 0.0f;
19            maxThreshold = 8.0f;
20        }
21        case ErrorCalculationMethod.SSIM → {
22            minThreshold = 0.0f;
23            maxThreshold = 1.0f;
24        }
25    }
26
27 System.out.println("Masukkan ambang batas kompresi.");
28 System.out.printf("Rentang untuk ambang batas kompresi: %.2f - %.2f\n", minThreshold, maxThreshold);
29 System.out.print("> ");
30
31 boolean valid = false;
32 while (!valid) {
33     String input = userInput.nextLine();
34     try {
35         float value = Float.parseFloat(input);
36
37         if (value < minThreshold || value > maxThreshold) {
38             System.out.printf("Nilai %.2f berada di luar rentang (%.2f - %.2f).\n", value, minThreshold, maxThreshold);
39             System.out.print("> ");
40             continue;
41         }
42
43         errorThreshold = value;
44         valid = true;
45     } catch (NumberFormatException e) {
46         System.out.println("Input tidak valid, silakan masukkan angka desimal!");
47         System.out.print("> ");
48     }
49 }
50 }
```

Method untuk menampilkan interface yang meminta error threshold manual dari pengguna. Hal-hal yang divalidasi adalah: range input (berdasarkan metode yang dipilih) dan tipe data masukkan.

```
● ● ●

1 public void getUserMinimumBlockSize(Scanner userInput) {
2     System.out.println("Masukkan luas blok minimum.");
3     System.out.print("> ");
4     boolean valid = false;
5     while (!valid) {
6         String input = userInput.nextLine();
7         try {
8             minimumBlockSize = Integer.parseInt(input);
9             if (minimumBlockSize > 0) {
10                 valid = true;
11             } else {
12                 System.out.println("Luas blok tidak boleh kurang dari 1.");
13             }
14         } catch (NumberFormatException e) {
15             System.out.println("Input tidak valid, silakan masukkan integer!");
16             System.out.print("> ");
17         }
18     }
19 }
```

Method untuk menampilkan interface yang meminta luas blok minimum dari pengguna. Hal-hal yang divalidasi adalah: range input (> 0) dan tipe data masukkan.



```
1 public void getUserImageOutputPath(Scanner userInput) {
2     System.out.println("Masukkan alamat absolut untuk gambar hasil kompresi.");
3     boolean valid = false;
4     while (!valid) {
5         System.out.print("> ");
6         String input = userInput.nextLine();
7         File outputFile = new File(input);
8         if (!outputFile.isAbsolute()) {
9             System.out.println("Masukkan alamat path absolut!");
10            continue;
11        }
12        File parentDir = outputFile.getParentFile();
13        if (parentDir != null && parentDir.exists() && parentDir.canWrite()) {
14            imageOutputPath = input;
15            valid = true;
16        } else {
17            System.out.println("Direktori tidak ada atau tidak memiliki akses. Silakan masukkan alamat yang valid!");
18        }
19    }
20 }
```

Method untuk menampilkan interface yang meminta alamat absolut gambar hasil kompresi (termasuk nama file gambar hasil). Hal-hal yang divalidasi adalah: apakah alamat absolut, dan akses ke direktori gambar(tidak ada atau tidak punya akses).

3.3.2. Quadtree

```
● ● ●  
1 private class Node {  
2     int x, y, width, height;  
3     Color averageColor;  
4     Node topLeft, topRight, bottomLeft, bottomRight;  
5     boolean isLeaf;  
6  
7     public Node(int x, int y, int width, int height) {  
8         this.x = x;  
9         this.y = y;  
10        this.width = width;  
11        this.height = height;  
12    }  
13}
```

Kelas untuk node pada pohon.

```
● ● ●  
1 private Node root;  
2 private final BufferedImage image;  
3 private final double errorThreshold;  
4 private final int minBlockSize;  
5 private final ErrorCalculationMethod method;  
6  
7 public Quadtree(BufferedImage image, double errorThreshold, int minBlockSize, ErrorCalculationMethod method) {  
8     this.image = image;  
9     this.errorThreshold = errorThreshold;  
10    this.minBlockSize = minBlockSize;  
11    this.method = method;  
12}
```

Atribut dan konstruktor

```
1 private Node buildTree(int x, int y, int width, int height) {
2     Node node = new Node(x, y, width, height);
3     double error = calculateError(x, y, width, height);
4
5     if (error < errorThreshold || width * height <= minBlockSize || width <= 1 || height <= 1) {
6         node.isLeaf = true;
7         node.averageColor = calculateAverageColor(x, y, width, height);
8     } else {
9         int halfWidth = Math.max(1, (width / 2));
10        int halfHeight = Math.max(1, (height / 2));
11        node.topLeft = buildTree(x, y, halfWidth, halfHeight);
12        if (width > halfWidth) {
13            node.topRight = buildTree(x + halfWidth, y, width - halfWidth, halfHeight);
14        }
15
16        if (height > halfHeight) {
17            node.bottomLeft = buildTree(x, y + halfHeight, halfWidth, height - halfHeight);
18        }
19
20        if (width > halfWidth && height > halfHeight) {
21            node.bottomRight = buildTree(x + halfWidth, y + halfHeight, width - halfWidth, height - halfHeight);
22        }
23    }
24    return node;
25 }
```

Method untuk membuat pohon secara rekursif. Base case berupa kalkulasi error atau luas blok minimum. Method akan terus membuat 4 node baru jika tidak masuk ke base case.

```
1 private Color calculateAverageColor(int x, int y, int width, int height) {
2     long sumR = 0, sumG = 0, sumB = 0;
3     int totalPixels = width * height;
4     for (int i = x; i < x + width; i++) {
5         for (int j = y; j < y + height; j++) {
6             int rgb = image.getRGB(i, j);
7             Color color = new Color(rgb);
8             sumR += color.getRed();
9             sumG += color.getGreen();
10            sumB += color.getBlue();
11        }
12    }
13    return new Color((int)(sumR/totalPixels), (int)(sumG/totalPixels), (int)(sumB/totalPixels));
14 }
```

Method untuk menghitung warna RGB rata-rata. Melakukan loop 2D untuk mencari total nilai RGB kemudian dibagi dengan luas blok (rata-rata).



```
● ● ●
1  private double calculateError(int x, int y, int width, int height) {
2      switch (method) {
3          case ErrorCalculationMethod.VARIANCE → {
4              return calculateErrorByVariance(x, y, width, height);
5          }
6          case ErrorCalculationMethod.MAD → {
7              return calculateErrorByMAD(x, y, width, height);
8          }
9          case ErrorCalculationMethod.MAX_PIXEL_DIFFERENCE → {
10             return calculateErrorByMPD(x, y, width, height);
11         }
12         case ErrorCalculationMethod.ENTROPY → {
13             return calculateErrorByEntropy(x, y, width, height);
14         }
15         case ErrorCalculationMethod.SSIM → {
16             return calculateErrorBySSIM(x, y, width, height);
17         }
18         default → {
19             return 0.0;
20         }
21     }
22 }
```

Menentukan method yang akan dipakai untuk perhitungan error.

```
● ● ●
1  private double calculateErrorByVariance(int x, int y, int width, int height) {
2      Color avgColor = calculateAverageColor(x, y, width, height);
3      double avgR = avgColor.getRed(), avgG = avgColor.getGreen(), avgB = avgColor.getBlue();
4
5      double sumR = 0.0, sumG = 0.0, sumB = 0.0;
6      int totalPixel = width * height;
7
8      for (int i = x; i < x + width; i++) {
9          for (int j = y; j < y + height; j++) {
10              int rgb = image.getRGB(i, j);
11              Color color = new Color(rgb);
12              double valueR = color.getRed() - avgR;
13              double valueG = color.getGreen() - avgG;
14              double valueB = color.getBlue() - avgB;
15              sumR += valueR * valueR;
16              sumG += valueG * valueG;
17              sumB += valueB * valueB;
18          }
19      }
20
21      double varR = sumR / totalPixel;
22      double varG = sumG / totalPixel;
23      double varB = sumB / totalPixel;
24
25      return (varR + varG + varB) / 3.0;
26  }
```

Method untuk menghitung error varians. Menghitung average color. Menghitung jumlah kuadrat selisih dengan average color. Lalu menghitung varians RGB.

```
1  private double calculateErrorByMAD(int x, int y, int width, int height) {  
2      Color avgColor = calculateAverageColor(x, y, width, height);  
3      double avgR = avgColor.getRed(), avgG = avgColor.getGreen(), avgB = avgColor.getBlue();  
4  
5      double valueR = 0.0, valueG = 0.0, valueB = 0.0;  
6      int totalPixel = width * height;  
7  
8      for (int i = x; i < x + width; i++) {  
9          for (int j = y; j < y + height; j++) {  
10              int rgb = image.getRGB(i, j);  
11              Color color = new Color(rgb);  
12              valueR += Math.abs(color.getRed() - avgR);  
13              valueG += Math.abs(color.getGreen() - avgG);  
14              valueB += Math.abs(color.getBlue() - avgB);  
15          }  
16      }  
17  
18      double madR = valueR / totalPixel;  
19      double madG = valueG / totalPixel;  
20      double madB = valueB / totalPixel;  
21  
22      return (madR + madG + madB) / 3.0;  
23  }
```

Method untuk menghitung error MAD. Menghitung average color. Menghitung jumlah mutlak selisih dengan average color. Lalu menghitung MPD RGB.

```
1  private double calculateErrorByMPD(int x, int y, int width, int height) {  
2      int maxR = 0, maxG = 0, maxB = 0;  
3      int minR = 255, minG = 255, minB = 255;  
4  
5      for (int i = x; i < x + width; i++) {  
6          for (int j = y; j < y + height; j++) {  
7              int rgb = image.getRGB(i, j);  
8              Color color = new Color(rgb);  
9              int r = color.getRed();  
10             int g = color.getGreen();  
11             int b = color.getBlue();  
12  
13             if (r > maxR) {  
14                 maxR = r;  
15             }  
16             if (g > maxG) {  
17                 maxG = g;  
18             }  
19             if (b > maxB) {  
20                 maxB = b;  
21             }  
22             if (r < minR) {  
23                 minR = r;  
24             }  
25             if (g < minG) {  
26                 minG = g;  
27             }  
28             if (b < minB) {  
29                 minB = b;  
30             }  
31         }  
32     }  
33  
34     double diffR = maxR - minR;  
35     double diffG = maxG - minG;  
36     double diffB = maxB - minB;  
37  
38     return (diffR + diffG + diffB) / 3.0;  
39 }
```

Method untuk menghitung error MPD. Loop 2D mencari nilai minimum dan maksimum. Menghitung selisih antara minimum dan maksimum. Lalu menghitung selisih RGB.

```
● ● ●
1 private double calculateErrorByEntropy(int x, int y, int width, int height) {
2     int[] histogram = new int[256];
3     for (int i = x; i < x + width; i++) {
4         for (int j = y; j < y + height; j++) {
5             int rgb = image.getRGB(i, j);
6             Color color = new Color(rgb);
7             int intensity = (color.getRed() + color.getGreen() + color.getBlue()) / 3;
8             histogram[intensity]++;
9         }
10    }
11
12    int totalPixels = width * height;
13    double entropy = 0.0;
14    for (int value : histogram) {
15        if (value > 0) {
16            double probability = (double) value / totalPixels;
17            entropy += -probability * (Math.log(probability) / Math.log(2));
18        }
19    }
20
21    return entropy;
22 }
```

Method untuk menghitung error entropi. Membuat dan mengisi histogram warna. Lalu, menghitung total entropi untuk setiap nilai histogram.

```

1  private double calculateErrorBySSIM(int x, int y, int width, int height) {
2      final double K1 = 0.01;
3      final double K2 = 0.03;
4      final double L = 255;
5      final double C1 = (K1 * L) * (K1 * L);
6      final double C2 = (K2 * L) * (K2 * L);
7
8      int count = width * height;
9      double sumR = 0.0, sumG = 0.0, sumB = 0.0;
10     double sumSquaredR = 0.0, sumSquaredG = 0.0, sumSquaredB = 0.0;
11
12     for (int i = x; i < x + width; i++) {
13         for (int j = y; j < y + height; j++) {
14             int rgb = image.getRGB(i, j);
15             Color color = new Color(rgb);
16             double r = color.getRed();
17             double g = color.getGreen();
18             double b = color.getBlue();
19             sumR += r;
20             sumG += g;
21             sumB += b;
22             sumSquaredR += r * r;
23             sumSquaredG += g * g;
24             sumSquaredB += b * b;
25         }
26     }
27
28     double muR = sumR / count;
29     double muG = sumG / count;
30     double muB = sumB / count;
31
32     double varR = (sumSquaredR / count) - (muR * muR);
33     double varG = (sumSquaredG / count) - (muG * muG);
34     double varB = (sumSquaredB / count) - (muB * muB);
35
36     double ssimR = C2 / (varR + C2);
37     double ssimG = C2 / (varG + C2);
38     double ssimB = C2 / (varB + C2);
39
40     double wR = 0.299;
41     double wG = 0.587;
42     double wB = 0.114;
43
44     return 1.0 - (wR * ssimR + wG * ssimG + wB * ssimB); // Invert for the error
45 }

```

Method untuk menghitung error SSIM. Menghitung C1 dan C2 dengan konstanta K1 = 0.01, K2 = 0.03, L = 255. Menghitung SSIM

dengan rumus yang telah disederhanakan. Mengembalikan nilai error dengan $1 - \text{SSIM}$ (karena SSIM menilai seberapa mirip).

```
● ● ●
1 public int getTreeDepth() {
2     int depth = countTreeDepth(root);
3     return depth > 0 ? depth - 1 : 0;
4 }
5
6 private int countTreeDepth(Node node) {
7     if (node == null) {
8         return 0;
9     }
10
11    if (node.isLeaf) {
12        return 1;
13    }
14
15    int depthTopLeft = countTreeDepth(node.topLeft);
16    int depthTopRight = countTreeDepth(node.topRight);
17    int depthBottomLeft = countTreeDepth(node.bottomLeft);
18    int depthBottomRight = countTreeDepth(node.bottomRight);
19    return 1 + (Math.max(Math.max(depthTopLeft, depthTopRight),
20                         Math.max(depthBottomLeft, depthBottomRight)));
21 }
```

Method untuk mencari kedalaman pohon.

1

```
public int getNodeCount() {  
    return countNodes(root);  
}  
  
private int countNodes(Node node) {  
    if (node == null) {  
        return 0;  
    }  
  
    int count = 1;  
    if(!node.isLeaf) {  
        count += countNodes(node.topLeft);  
        count += countNodes(node.topRight);  
        count += countNodes(node.bottomLeft);  
        count += countNodes(node.bottomRight);  
    }  
    return count;  
}
```

Method untuk mencari banyak node.

```
● ● ●
1 public BufferedImage getCompressedImage() {
2     BufferedImage output = new BufferedImage(image.getWidth(), image.getHeight(), BufferedImage.TYPE_INT_RGB);
3     drawTree(root, output);
4     return output;
5 }
6
7 private void drawTree(Node node, BufferedImage output) {
8     if (node == null) {
9         return;
10    }
11    if (node.isLeaf) {
12        for (int i = node.x; i < node.x + node.width; i++) {
13            for (int j = node.y; j < node.y + node.height; j++) {
14                output.setRGB(i, j, node.averageColor.getRGB());
15            }
16        }
17        return;
18    }
19    drawTree(node.topLeft, output);
20    drawTree(node.topRight, output);
21    drawTree(node.bottomLeft, output);
22    drawTree(node.bottomRight, output);
23 }
```

Method untuk menggambar dan mengembalikan gambar.

3.3.1. Main

```
● ● ●
1 InputManager inputManager = new InputManager();
2 inputManager.getUserImage(scanner);
3 inputManager.getUserErrorMethod(scanner);
4 inputManager.getUserMinimumCompressionPercentage(scanner);
5 if (inputManager.getMinCompressionPercentage() > 0.0) {
6     inputManager.getUserMaxSearchAttempts(scanner);
7 }
8 if (inputManager.getMinCompressionPercentage() ≤ 0.0) {
9     inputManager.getUserThreshold(scanner);
10}
11 inputManager.getUserMinimumBlockSize(scanner);
12 inputManager.getUserImageOutputPath(scanner);
```

Bagian yang menggunakan objek InputManager untuk masukkan pengguna.

```
● ● ●
1 long startTime = System.currentTimeMillis();
2 Quadtree quadtree = new Quadtree(inputManager.getImage(), inputManager.getErrorThreshold(), inputManager.getMinimumBlockSize(), inputManager.getMethod());
3 quadtree.build();
4
5 long endTime = System.currentTimeMillis();
6 long executionTime = endTime - startTime;
```

Bagian untuk membuat pohon kompresi gambar. Dilakukan juga penghitungan waktu.

```
● ● ●
1 BufferedImage compressedImage = quadtree.getCompressedImage();
2 String outputPath = inputManager.getImageOutputPath();
3 String extension = outputPath.substring(outputPath.lastIndexOf('.') + 1).toLowerCase();
4
5 File outputFile = new File(outputPath);
6 try {
7     ImageIO.write(compressedImage, extension, outputFile);
8 } catch (IOException e) {
9     System.out.println("Terjadi kesalahan saat menyimpan gambar: " + e.getMessage());
10 }
```

Bagian untuk menyimpan gambar hasil kompresi.

```
1 long compressedSize = outputFile.length();
2 int treeDepth = quadtree.getTreeDepth();
3 int nodeCount = quadtree.getNodeCount();
4 double compressionPercentage = (1 - ((double) compressedSize / originalSize)) * 100;
5 double compressionRatio = (double) originalSize / compressedSize;
6
7 System.out.println("\n===== COMPRESSION STATISTICS =====");
8 System.out.println("Waktu eksekusi : " + executionTime + " ms");
9 System.out.println("Ukuran gambar original : " + formatFileSize(originalSize));
10 System.out.println("Ukuran gambar hasil kompresi : " + formatFileSize(compressedSize));
11 System.out.printf("Rasio kompresi : %.2f:1\n", compressionRatio);
12 System.out.printf("Persentase kompresi : %.2f%\n", compressionPercentage);
13 System.out.println("Kedalaman pohon : " + treeDepth);
14 System.out.println("Banyak node : " + nodeCount);
```

Bagian untuk menghitung statistik dan menampilkannya.

```
1 private static String formatFileSize(long size) {
2     final String[] units = new String[] {"B", "KB", "MB", "GB"};
3     int unitIndex = 0;
4     double dataSize = size;
5
6     while (dataSize >= 1024 && unitIndex < units.length - 1) {
7         dataSize /= 1024;
8         unitIndex++;
9     }
10
11    return String.format("%.2f %s", dataSize, units[unitIndex]);
12 }
```

Fungsi helper untuk output statistik. Menampilkan ukuran file dengan satuan yang tepat.

```
● ● ●
1 private static void findOptimalParameters(InputManager inputManager, long originalSize) throws IOException {
2     System.out.println("\nMencari parameter optimal...");
3
4     double targetCompressionPercentage = inputManager.getMinCompressionPercentage();
5     ErrorCalculationMethod method = inputManager.getMethod();
6
7     double minThreshold = getMinThresholdForMethod(method);
8     double maxThreshold = getMaxThresholdForMethod(method);
9     double range = maxThreshold - minThreshold;
10
11    double initialPercentageRange = 0.02;
12    double currentThreshold = minThreshold + (maxThreshold - minThreshold) * initialPercentageRange;
13    double currentCompressionPercentage = 0;
14    double previousThreshold = currentThreshold;
15
16    int maxAttempts = inputManager.getMaxSearchAttempts();
17    boolean targetReached = false;
```

Fungsi untuk mencari error threshold paling mendekati dan di atas target threshold. Bagian satu ini berisi variabel yang digunakan.

```
● ● ●
1  for (int attempt = 0; attempt < maxAttempts; attempt++) {
2      if (attempt > 0) {
3          double increment;
4
5          if (range < 10) {
6              increment = 1.3;
7          } else if (range < 1000) {
8              increment = 1.6;
9          } else {
10              increment = 2.0;
11          }
12
13      boolean isAboveTarget = currentCompressionPercentage ≥ targetCompressionPercentage;
14
15      if (!isAboveTarget) {
16          double diff = targetCompressionPercentage - currentCompressionPercentage;
17          if (diff > 0.3) {
18              increment *= 2;
19          } else if (diff < 0.1) {
20              increment = 1.1;
21          }
22      } else {
23          double diff = currentCompressionPercentage - targetCompressionPercentage;
24          if (diff > 0.3) {
25              increment = 0.5;
26          } else if (diff > 0.1) {
27              increment = 0.7;
28          } else {
29              increment = 0.9;
30          }
31      }
32
33      previousThreshold = currentThreshold;
34      currentThreshold = Math.min(currentThreshold * increment, maxThreshold);
35
36  }
```

Bagian 2 lanjutan sebelumnya. Melakukan perubahan terhadap error threshold berdasarkan selisih persentase saat ini dan target

```
● ● ●
1 Quadtree test = new Quadtree(inputManager.getImage(), currentThreshold, inputManager.getMinimumBlockSize(), method);
2 test.build();
3
4 BufferedImage compressedImage = test.getCompressedImage();
5
6 ByteArrayOutputStream outputStream = new ByteArrayOutputStream();
7 String extension = inputManager.getImageOutputPath().substring(inputManager.getImageOutputPath().lastIndexOf('.') + 1).toLowerCase();
8
9 ImageIO.write(compressedImage, extension, outputStream);
10 long estimatedSize = outputStream.size();
11
12 currentCompressionPercentage = 1.0 - ((double)estimatedSize / originalSize);
```

Bagian 3 lanjutan sebelumnya. Melakukan kompresi dan mencari persentase kompresi saat ini.

```
● ● ●
1     if (currentCompressionPercentage > bestCompressionPercentage) {
2         bestCompressionPercentage = currentCompressionPercentage;
3         bestThreshold = currentThreshold;
4     }
5
6     if (currentCompressionPercentage ≥ targetCompressionPercentage) {
7         targetReached = true;
8
9         if (currentCompressionPercentage < bestTargetCompressionPercentage) {
10             bestTargetCompressionPercentage = currentCompressionPercentage;
11             bestTargetThreshold = currentThreshold;
12         }
13     }
14
15     if (attempt > 0 && currentCompressionPercentage < bestCompressionPercentage * 0.08) {
16         currentThreshold = (previousThreshold + currentThreshold) / 2;
17     }
18 }
```

Menentukan error threshold saat ini yang paling baik.

```
1 public enum ErrorCalculationMethod {  
2     VARIANCE,  
3     MAD,  
4     MAX_PIXEL_DIFFERENCE,  
5     ENTROPY,  
6     SSIM  
7 }
```

Enumerasi untuk tipe metode kalkulasi error.

BAB IV

EKSPERIMENT DAN ANALISIS

4.1. Pengujian Program

Semua foto untuk pengujian berada di folder test/input dan output di folder test/output. Berikut ini adalah beberapa hasil uji dengan minimum block size dibuat selalu sama (1).

Gambar Masukkan 1 (2560x1440, sumber:

<https://www.wallpaperflare.com/man-near-torii-gate-wallpaper-gray-temple-wallpaper-landscape-wallpaper-cqg>)



Uji 1: Metode varians, threshold 6502.5 (40% maks), min block size 1



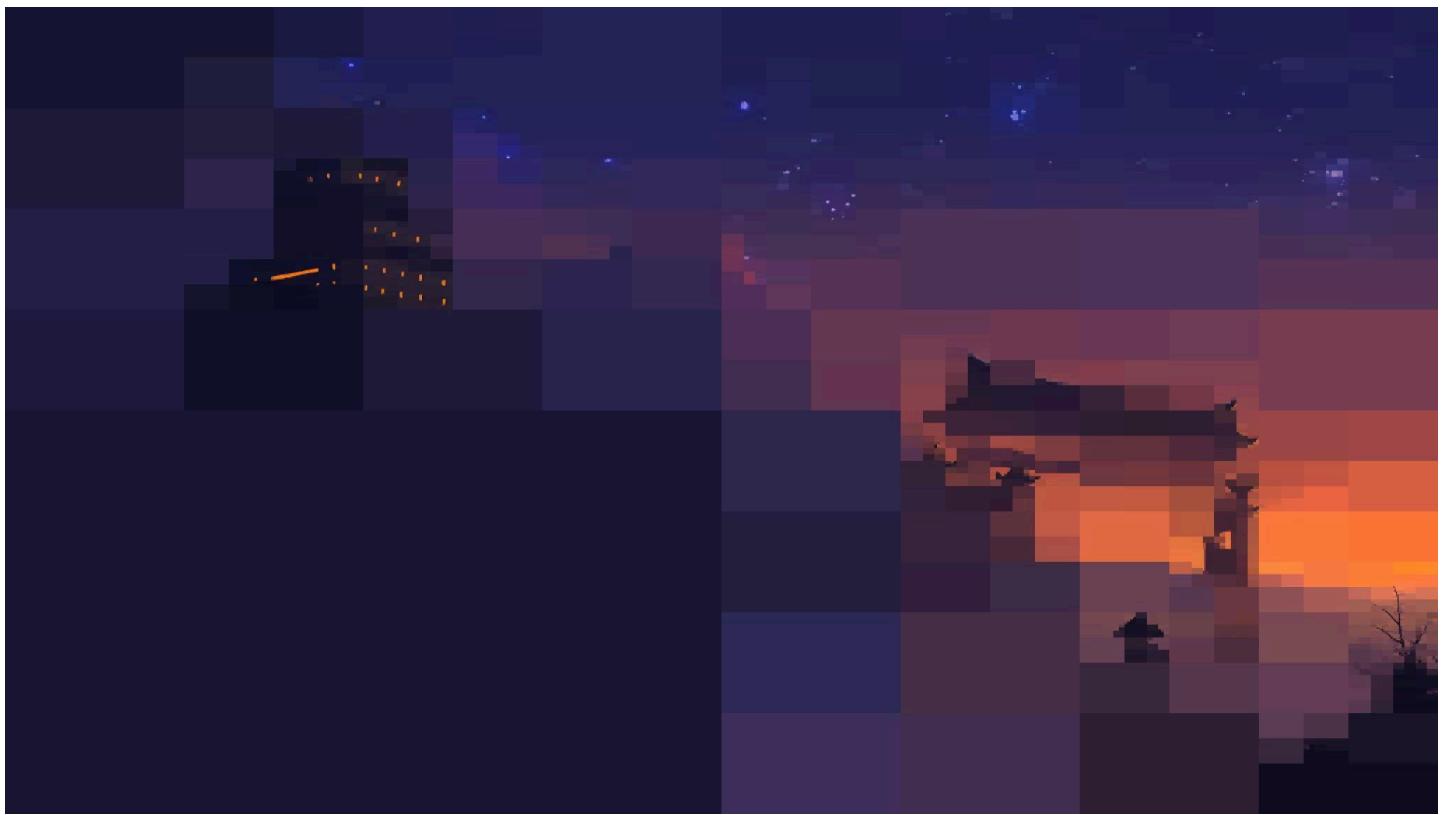
```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 291 ms
Ukuran gambar original : 232.07 KB
Ukuran gambar hasil kompresi : 56.86 KB
Rasio kompresi : 4.08:1
Persentase kompresi : 75.50%
Kedalaman pohon : 0
Banyak node : 1
```

Uji 2: Metode MAD, threshold 51 (40% maks), min block size 1



```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 291 ms
Ukuran gambar original : 232.07 KB
Ukuran gambar hasil kompresi : 56.86 KB
Rasio kompresi : 4.08:1
Persentase kompresi : 75.50%
Kedalaman pohon : 0
Banyak node : 1
```

Uji 3: Metode MPD, threshold 102 (40% maks), min block size 1



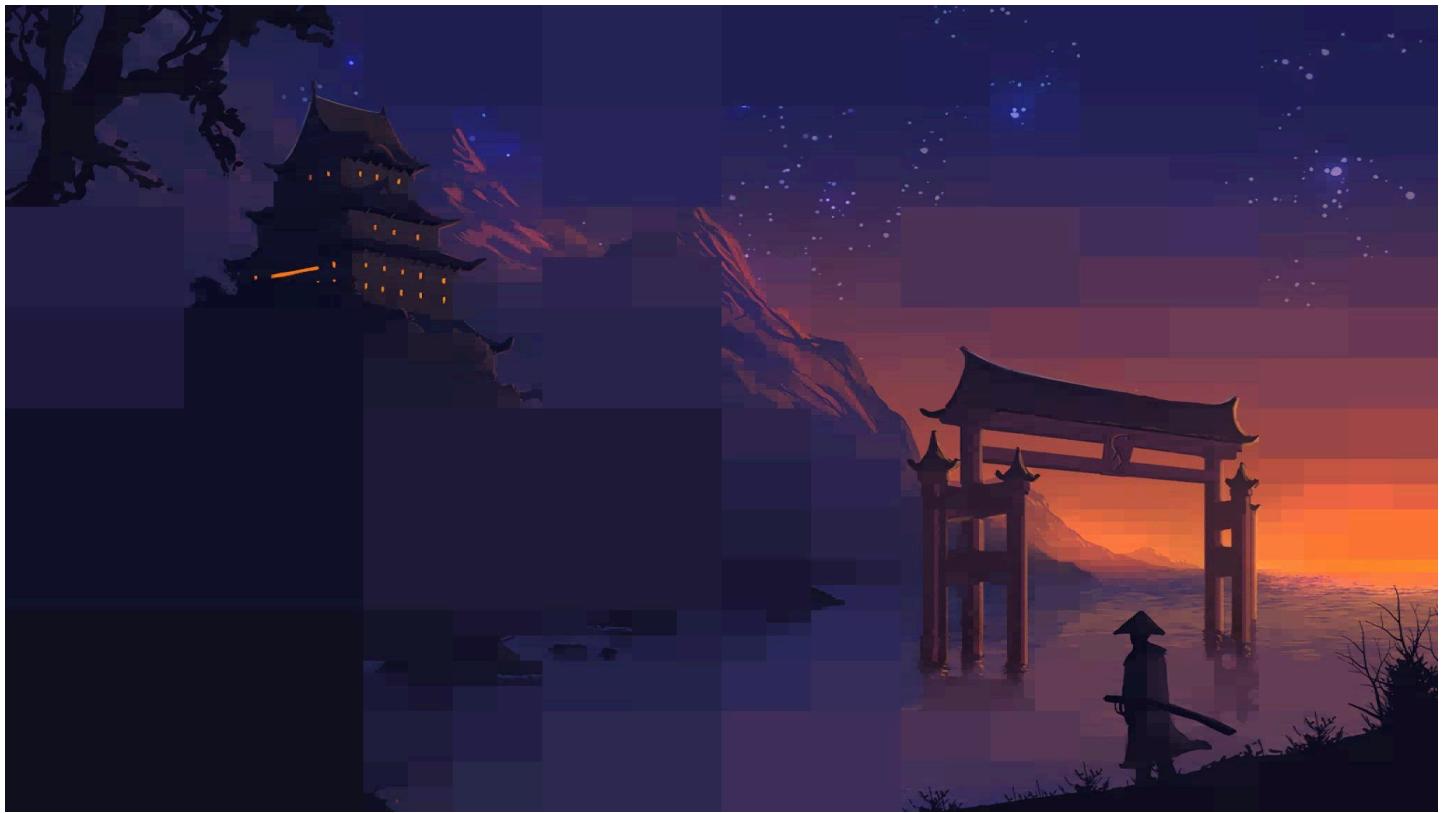
```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 524 ms
Ukuran gambar original : 232.07 KB
Ukuran gambar hasil kompresi : 83.27 KB
Rasio kompresi : 2.79:1
Persentase kompresi : 64.12%
Kedalaman pohon : 11
Banyak node : 4173
```

Uji 4: Metode entropi, threshold 3.2 (40% maks), min block size 1



```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 711 ms
Ukuran gambar original : 232.07 KB
Ukuran gambar hasil kompresi : 144.68 KB
Rasio kompresi : 1.60:1
Persentase kompresi : 37.66%
Kedalaman pohon : 10
Banyak node : 82229
```

Uji 5: Metode SSIM, threshold 0.4 (40% maks), min block size 1



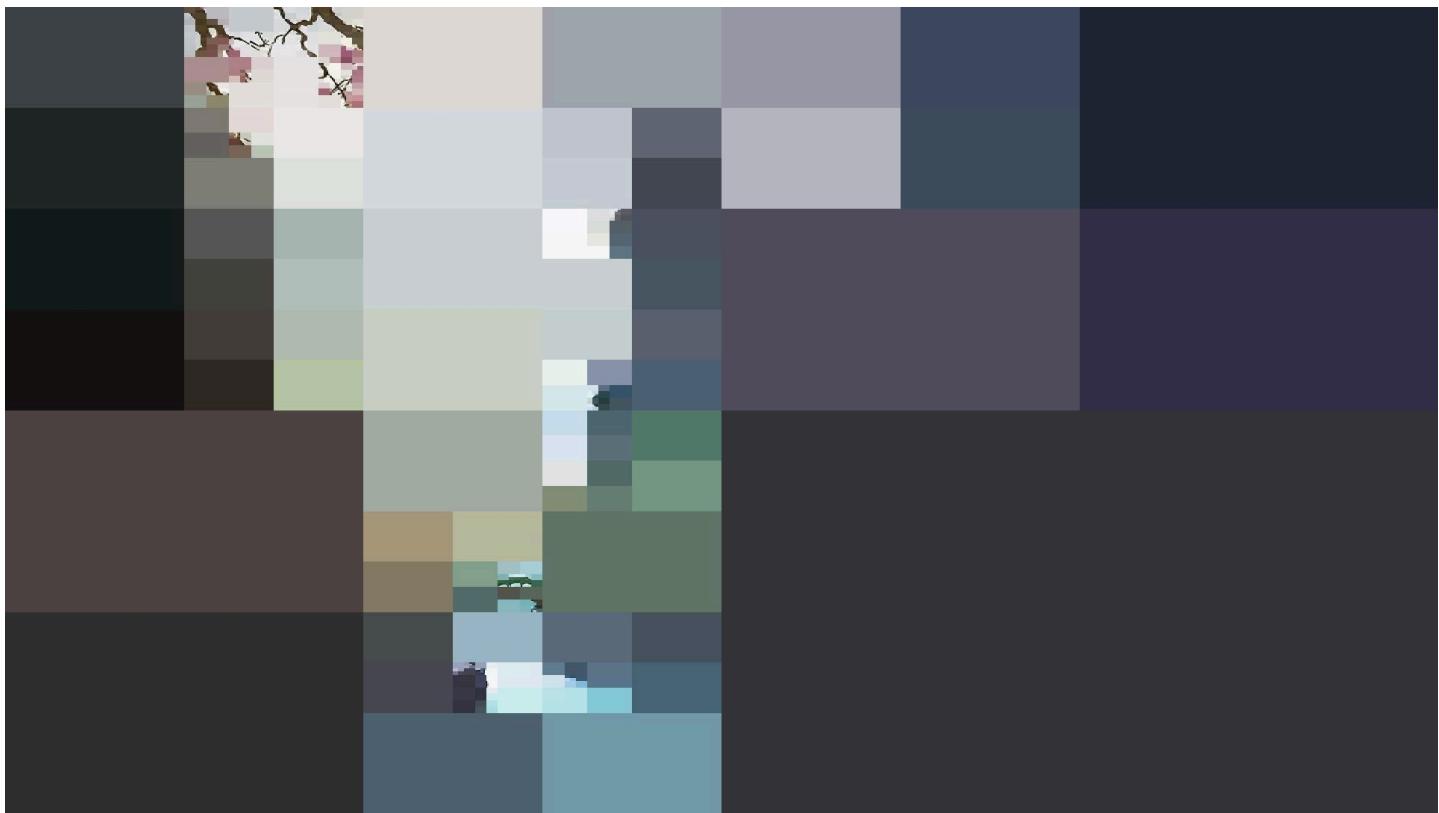
```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 592 ms
Ukuran gambar original : 232.07 KB
Ukuran gambar hasil kompresi : 128.96 KB
Rasio kompresi : 1.80:1
Persentase kompresi : 44.43%
Kedalaman pohon : 11
Banyak node : 95053
```

Gambar Masukkan 2 (1920x1080, sumber:

<https://www.wallpaperflare.com/cherry-blossoms-tree-near-waterfall-wallpaper-woodl-and-stream-illustration-wallpaper-qwp>)

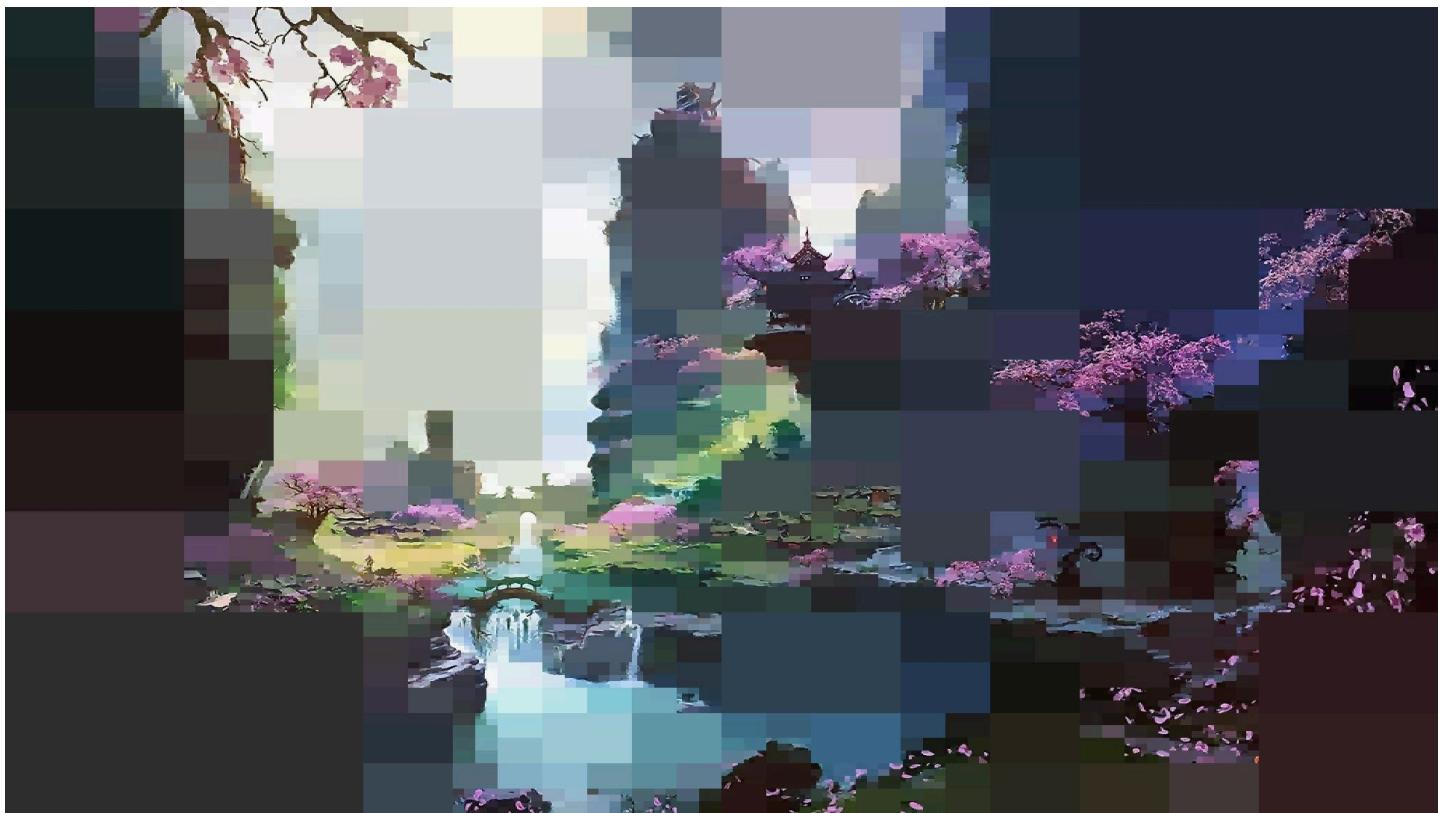


Uji 6: Metode varians, threshold 3251.25 (20% maks), min block size 1



```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 395 ms
Ukuran gambar original : 554.99 KB
Ukuran gambar hasil kompresi : 50.44 KB
Rasio kompresi : 11.00:1
Persentase kompresi : 90.91%
Kedalaman pohon : 11
Banyak node : 2569
```

Uji 7: Metode MAD, threshold 25.5 (20% maks), min block size 1



```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 638 ms
Ukuran gambar original : 554.99 KB
Ukuran gambar hasil kompresi : 156.65 KB
Rasio kompresi : 3.54:1
Persentase kompresi : 71.77%
Kedalaman pohon : 11
Banyak node : 68681
```

Uji 8: Metode MPD, threshold 51 (20% maks), min block size 1



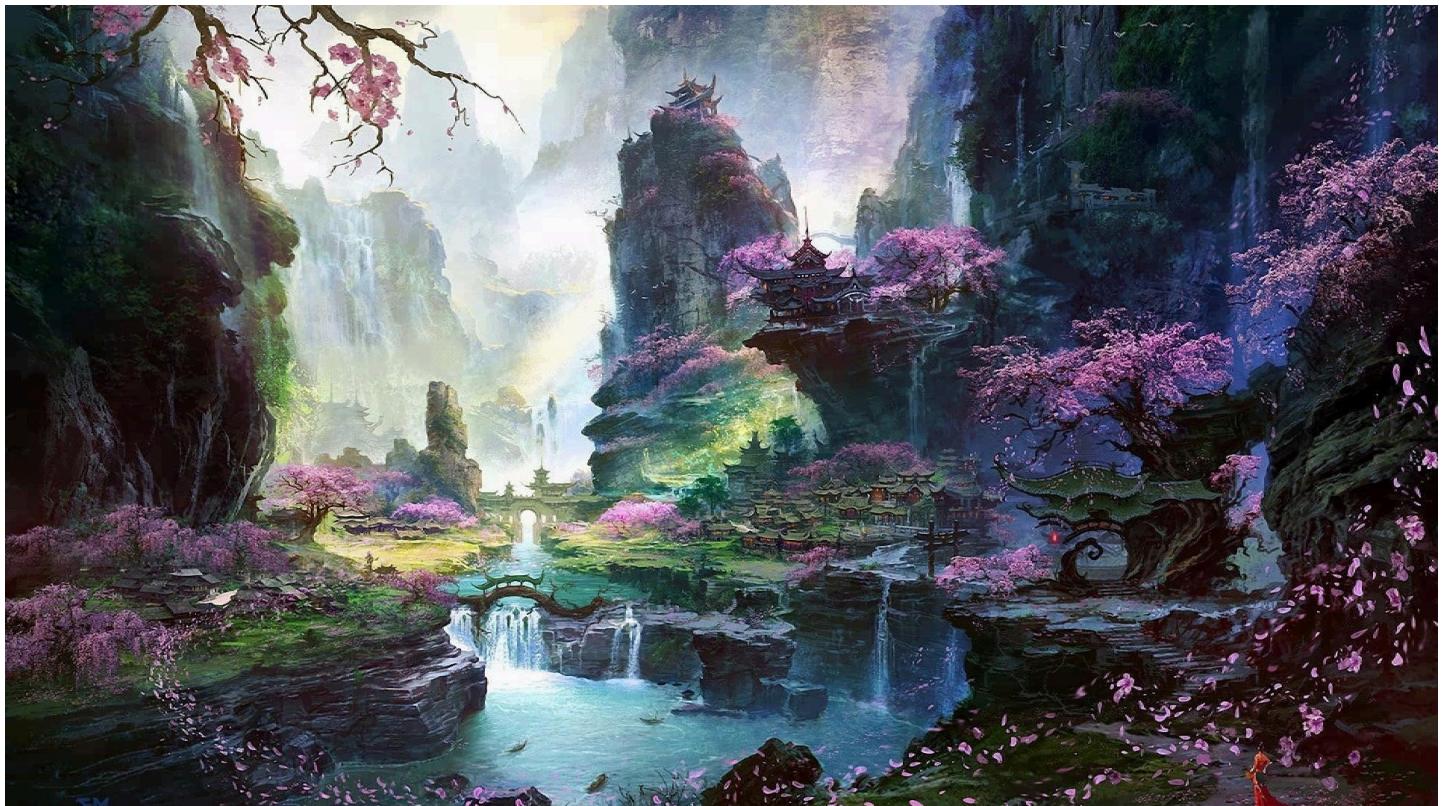
```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 583 ms
Ukuran gambar original : 554.99 KB
Ukuran gambar hasil kompresi : 360.31 KB
Rasio kompresi : 1.54:1
Persentase kompresi : 35.08%
Kedalaman pohon : 11
Banyak node : 408729
```

Uji 9: Metode entropi, threshold 1.6 (20% maks), min block size 1



```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 1190 ms
Ukuran gambar original : 554.99 KB
Ukuran gambar hasil kompresi : 441.62 KB
Rasio kompresi : 1.26:1
Persentase kompresi : 20.43%
Kedalaman pohon : 11
Banyak node : 1515801
```

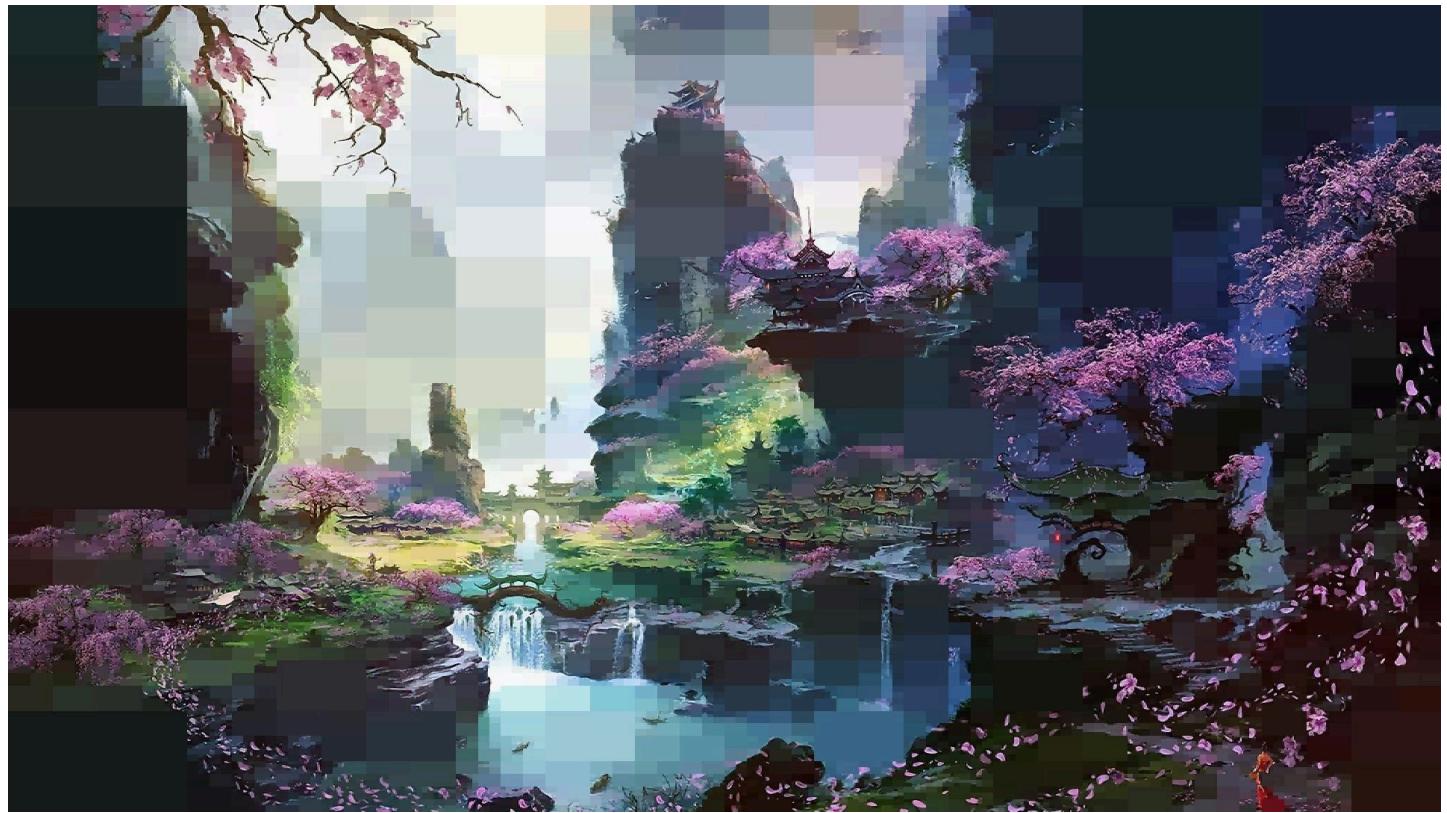
Uji 10: Metode SSIM, threshold 0.2 (20% maks), min block size 1



===== COMPRESSION STATISTICS =====

Waktu eksekusi	: 664 ms
Ukuran gambar original	: 554.99 KB
Ukuran gambar hasil kompresi	: 432.08 KB
Rasio kompresi	: 1.28:1
Persentase kompresi	: 22.15%
Kedalaman pohon	: 11
Banyak node	: 1247885

Uji 11: Metode Kompresi SSIM, target kompresi 50%, percobaan 50x, minimum block size 1



Berhasil menemukan parameter yang memenuhi target kompresi.
Menggunakan parameter: Threshold = 0.87, Block Size = 1

```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 519 ms
Ukuran gambar original : 554.99 KB
Ukuran gambar hasil kompresi : 276.99 KB
Rasio kompresi : 2.00:1
Persentase kompresi : 50.09%
Kedalaman pohon : 11
Banyak node : 245197
- Error threshold : 0.8724774
```

Parameter ditemukan pada percobaan ke-34

```
Percobaan 30 / 50: Threshold = 0.67 (67.0% dari maksimum)
Hasil: Kompresi 34.46% (Target: 50.00%)
```

```
Percobaan 31 / 50: Threshold = 0.87 (87.2% dari maksimum)
Hasil: Kompresi 49.99% (Target: 50.00%)
```

```
Percobaan 32 / 50: Threshold = 0.96 (95.9% dari maksimum)
Hasil: Kompresi 78.79% (Target: 50.00%)
```

```
Percobaan 33 / 50: Threshold = 0.67 (67.1% dari maksimum)
Hasil: Kompresi 34.47% (Target: 50.00%)
```

```
Percobaan 34 / 50: Threshold = 0.87 (87.2% dari maksimum)
Hasil: Kompresi 50.09% (Target: 50.00%)
```

```
Percobaan 35 / 50: Threshold = 0.79 (78.5% dari maksimum)
Hasil: Kompresi 41.18% (Target: 50.00%)
```

BONUS Gambar Kucing (transparan, 1261x1715, sumber: arsip penulis)



Uji 12: Metode Kompresi entropi, target kompresi 40%, percobaan 50, minimum block size 1



Berhasil menemukan parameter yang memenuhi target kompresi.
Menggunakan parameter: Threshold = 2.23, Block Size = 1

```
===== COMPRESSION STATISTICS =====
Waktu eksekusi : 562 ms
Ukuran gambar original : 1.02 MB
Ukuran gambar hasil kompresi : 626.69 KB
Rasio kompresi : 1.67:1
Persentase kompresi : 40.18%
Kedalaman pohon : 10
Banyak node : 248153
- Error threshold : 2.2316341
```

Parameter ditemukan pada percobaan ke-30

Percobaan 25 / 50: Threshold = 2.07 (25.9% dari maksimum)
Hasil: Kompresi 35.33% (Target: 40.00%)

Percobaan 26 / 50: Threshold = 2.28 (28.5% dari maksimum)
Hasil: Kompresi 44.08% (Target: 40.00%)

Percobaan 27 / 50: Threshold = 2.05 (25.6% dari maksimum)
Hasil: Kompresi 34.84% (Target: 40.00%)

Percobaan 28 / 50: Threshold = 2.25 (28.2% dari maksimum)
Hasil: Kompresi 43.95% (Target: 40.00%)

Percobaan 29 / 50: Threshold = 2.03 (25.4% dari maksimum)
Hasil: Kompresi 34.66% (Target: 40.00%)

Percobaan 30 / 50: Threshold = 2.23 (27.9% dari maksimum)
Hasil: Kompresi 40.18% (Target: 40.00%)

4.2. Analisis Algoritma

Quadtree merupakan algoritma divide and conquer dengan membagi suatu masalah menjadi bentuk terkecilnya, menyelesaikannya satu per satu, dan menggabungkan solusi-solusi tersebut. Quadtree compression mengikuti suatu pola yaitu dekomposisi rekursi secara top-down. Algoritma ini berawal dari satu gambar keseluruhan kemudian membagi gambar ini menjadi area yang lebih kecil hingga batas tertentu.

Pencarian kompleksitas waktu untuk algoritma ini memerlukan analisis terhadap setiap proses yang dilakukan. Proses pertama yang dilakukan adalah pengecekan error. Untuk metode varians, pertama dilakukan pencarian nilai rata-rata warna pada area tersebut. Algoritma ini akan melakukan traversal seluruh piksel sehingga memiliki kompleksitas $O(m \times n)$ untuk m adalah lebar dan n adalah tinggi area. Kemudian metode ini akan melakukan traversal sekali lagi untuk mencari kuadrat selisih sehingga kompleksitas waktunya adalah

$$O(m \times n) + O(m \times n) = O(m \times n)$$

Kompleksitas ini juga berlaku untuk metode MAD. Untuk metode MPD, algoritma ini juga harus melakukan traversal seluruh piksel pada area untuk mencari nilai minimum dan maksimum sehingga juga memiliki kompleksitas $O(m \times n)$. Metode entropi juga melakukan traversal penuh untuk mengisi nilai histogram lalu melakukan kalkulasi untuk setiap nilai pada histogram sehingga juga memiliki kompleksitas $O(m \times n)$. Terakhir, metode SSIM mirip seperti varians dan MAD juga mencari nilai rata-rata piksel sehingga memiliki kompleksitas yang sama.

Setelah menghitung error, program akan membagi area gambar menjadi 4 node dan proses ini memiliki kompleksitas

konstan $O(1)$. Proses ini akan terus berulang hingga, pada kasus terburuk, quadtree memiliki node sebanyak jumlah piksel. Kasus terburuk ini akan terjadi dengan kompleksitas waktu

$$O(\log_4(m \times n))$$

atau jika $m = n$

$$O(\log_4(n^2)) = O(\log(n))$$

Karena pengecekan error terjadi setiap kali gambar dibagi menjadi 4, total kompleksitas waktu terburuk adalah

$$O(m \times n) \times O(\log_4(m \times n)) = O(m \times n \cdot \log_4(m \times n))$$

atau jika $m = n$

$$O(n^2 \cdot \log(n))$$

Untuk kasus terbaik, algoritma ini hanya menghasilkan satu node sehingga kompleksitas waktunya adalah

$$\Omega(n^2)$$

Kemudian untuk memperoleh hasil kompresi, algoritma pasti melakukan traversal untuk menggambar tiap piksel sehingga memiliki kompleksitas waktu

$$\Theta(n^2)$$

Perhitungan error juga dapat dituliskan dengan big-theta notation karena mereka juga memiliki upper-bound dan lower-bound yang sama.

Untuk algoritma pencarian target kompresi, algoritma akan terus diulang sesuai dengan input dari pengguna (k). Sehingga total kompleksitas waktunya adalah

$$O(k \cdot m \times n \cdot \log_4(m \times n))$$

atau

$$O(k \cdot n^2 \cdot \log(n))$$

BAB V

LAMPIRAN

No	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		✓
9	Program dibuat oleh saya sendiri	✓	

Repositori Program: https://github.com/V-Kleio/Tucil2_13523146

Pustaka

- Dokumen spesifikasi tugas kecil 2 oleh asisten IF2211 Strategi Algoritma Semester II tahun 2024/2025
- <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>
- <https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics/>
- <https://ieeexplore.ieee.org/document/1284395>
- <https://www.wallpaperflare.com/man-near-torii-gate-wallpaper-gray-temple-wallpaper-landscape-wallpaper-cqg>
- <https://www.wallpaperflare.com/cherry-blossoms-tree-near-waterfall-wallpaper-woodland-stream-illustration-wallpaper-qwp>
- <https://www.techtarget.com/whatis/definition/image-compression>