

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:

Nathanael Rachmat (13523142)

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

**BANDUNG
2025**

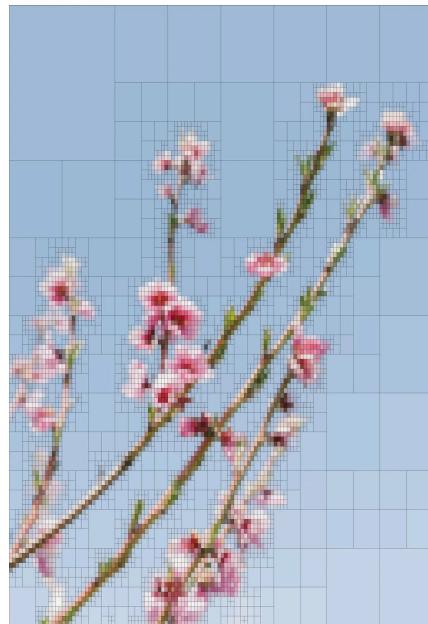
DAFTAR ISI

DAFTAR ISI.....	2
BAB I.....	4
BAB II.....	6
2.1. Divide and Conquer.....	6
2.2. Quadtree.....	7
2.3. Error Measurement Methods.....	7
2.3.1 Variance.....	7
2.3.1 Mean Absolute Deviation (MAD).....	8
2.3.1 Max Pixel Difference (MPD).....	8
2.3.1 Entropy.....	9
BAB III.....	10
3.1. Basis Rekursi.....	10
3.2. Pembagian Blok (Divide).....	10
3.3. Pemanggilan Rekursif (Conquer).....	10
3.4. Penggabungan Sub-Blok (Merge).....	11
3.5. Evaluasi Homogenitas Blok (Heuristik).....	11
BAB IV.....	12
4.1. Struktur Direktori.....	12
4.1.1 Direktori src/.....	12
4.1.2 Direktori bin/.....	13
4.1.3 Direktori doc/.....	13
4.1.4 Direktori test/.....	13
4.1.5 Direktori pendukung.....	13
4.2. App Class.....	13
4.2.1 Method Select Image File.....	14
4.2.2 Method Print.....	14
4.2.3 Method Choose Compressor.....	14
4.2.4 Close and Main Method.....	15
4.3. Input Scanner Class.....	15
4.4. Compress Quadtree class.....	16
4.4.1 Constructor.....	16
4.4.2 Helper Methods.....	17
4.4.3 Abstract Quad Node and Quad Builder Class.....	17
4.5. Compress Variance Class.....	18
4.5.1 Quad Node Variance Class.....	18
4.5.2 Quad Builder Variance Class.....	19
4.5.3 Calculate Variance.....	19
4.6. Compress MAD Class.....	20
4.6.1 Quad Node MAD Class.....	20
4.6.2 Quad Builder MAD Class.....	21
4.6.3 Calculate MAD.....	22
4.7. Compress MPD class.....	23
4.7.1 Quad Node MPD Class.....	23

4.7.2 Quad Builder MPD Class.....	24
4.7.3 Calculate Max Difference.....	25
4.8. Compress Entropy Class.....	25
4.8.1 Quad Node Entropy Class.....	26
4.8.2 Quad Builder Entropy Class.....	27
4.8.3 Calculate Entropy Class.....	28
BAB V.....	29
5.1. Pengujian Program.....	29
5.1.1 Pengujian 1.....	29
5.1.2 Pengujian 2.....	29
5.1.3 Pengujian 3.....	30
5.1.4 Pengujian 4.....	30
5.1.5 Pengujian 5.....	31
5.1.6 Pengujian 6.....	31
5.1.7 Pengujian 7.....	32
5.1.7 Pengujian 8.....	32
5.2. Analisis Hasil Percobaan.....	33
5.2.1 Analisis Algoritma Metode Variance, MPD, dan Entropy.....	33
5.2.2 Analisis Algoritma Metode MAD.....	34
LAMPIRAN.....	36

BAB I

DESKRIPSI MASALAH



Gambar 1.1. 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.



Gambar 1.2. Proses Pembentukan Quadtree dalam Kompresi Gambar

(Sumber: https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif)

Untuk tugas ini, diminta 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 berikut sebagai user input:

1. [INPUT] Alamat absolut gambar yang akan dikompresi.
2. [INPUT] Metode perhitungan galat (dengan penomoran sebagai input).
3. [INPUT] Ambang batas sesuai range dari metode yang dipilih.
4. [INPUT] Ukuran blok (simpul quadtree) minimum.
5. [INPUT] Target persentase kompresi dengan range 0.0 - 1.0. Nilai 0 berarti mode dinonaktifkan. Jika mode aktif, maka nilai ambang batas akan disesuaikan secara otomatis dan ukuran blok minimum tidak berlaku.
6. [INPUT] Alamat absolut gambar hasil kompresi.
7. [INPUT] Alamat absolut GIF.
8. [OUTPUT] Waktu eksekusi.
9. [OUTPUT] Ukuran gambar sebelum kompresi.
10. [OUTPUT] Ukuran gambar sesudah kompresi.
11. [OUTPUT] Persentase kompresi.
12. [OUTPUT] Kedalaman pohon.
13. [OUTPUT] Banyak simpul pada pohon,
14. [OUTPUT] Gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi (looping) pada alamat yang sudah ditentukan.

BAB II

TEORI SINGKAT

2.1. Divide and Conquer

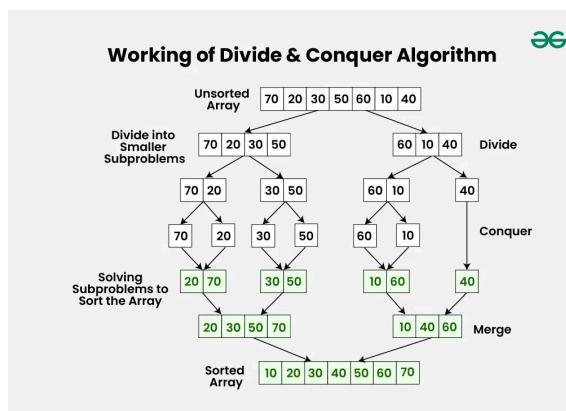
Algoritma Divide and Conquer adalah teknik pemecahan masalah yang membagi suatu masalah menjadi submasalah yang lebih kecil, menyelesaikan masing-masing secara independen, dan kemudian menggabungkan hasilnya untuk mendapatkan solusi dari masalah asli. Proses ini melibatkan tiga langkah utama:

1. **Divide (Membagi):** Memecah masalah utama menjadi beberapa submasalah yang lebih kecil.
2. **Conquer (Menaklukkan):** Menyelesaikan setiap submasalah secara rekursif hingga mencapai kasus dasar yang cukup sederhana untuk diselesaikan secara langsung.
3. **Combine (Menggabungkan):** Menggabungkan solusi dari submasalah untuk membentuk solusi akhir dari masalah utama.

Contoh algoritma yang menggunakan pendekatan ini meliputi:

- **Merge Sort:** Mengurutkan array dengan membaginya menjadi dua bagian, mengurutkan setiap bagian secara rekursif, dan kemudian menggabungkannya kembali.
- **Quick Sort:** Memilih elemen pivot, membagi array berdasarkan pivot, dan kemudian mengurutkan bagian-bagian tersebut secara rekursif.
- **Binary Search:** Mencari elemen dalam array yang sudah diurutkan dengan membagi ruang pencarian menjadi dua hingga menemukan elemen yang dicari.

Keuntungan utama dari algoritma Divide and Conquer adalah efisiensinya dalam menyelesaikan masalah kompleks dengan waktu komputasi yang lebih baik. Namun, algoritma ini juga memiliki kelemahan, seperti overhead rekursi dan kebutuhan memori tambahan untuk menyimpan hasil submasalah.



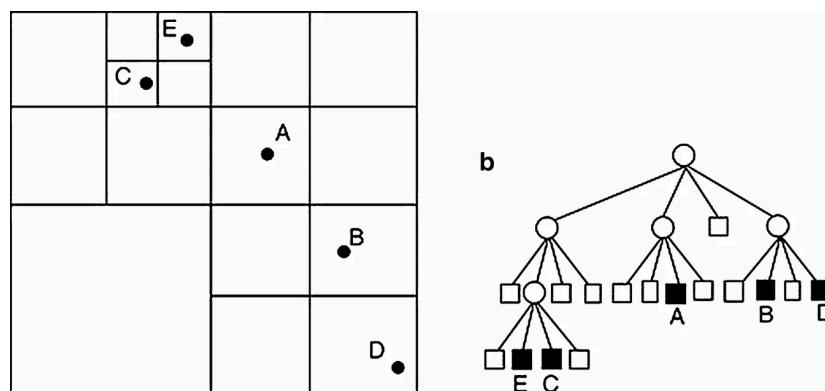
Gambar 2.1. Contoh *merge sort* dengan algoritma *divide and conquer*

(Sumber: <https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/>)

2.2. Quadtree

Quadtree adalah struktur data pohon di mana setiap simpul memiliki empat anak, digunakan untuk membagi ruang dua dimensi menjadi sub-ruang yang lebih kecil. Dalam kompresi gambar, quadtree bekerja dengan membagi gambar secara rekursif menjadi empat bagian. Setiap bagian menyimpan nilai warna rata-rata (RGB) dan kesalahan (error) dari rata-rata tersebut. Jika kesalahan melebihi ambang batas yang ditentukan, bagian tersebut dibagi lagi menjadi empat sub-bagian. Proses ini berlanjut hingga kesalahan dalam setiap bagian berada di bawah ambang batas atau ukuran bagian mencapai batas minimum.

Pendekatan ini memungkinkan representasi gambar yang lebih efisien dengan menyimpan lebih banyak detail di area yang kompleks dan lebih sedikit detail di area yang seragam. Selain mengurangi ukuran penyimpanan, quadtree juga mempercepat proses seperti deteksi tepi, karena hanya perlu memproses simpul daun dan induknya. Dengan demikian, quadtree menjadi metode kompresi gambar yang efektif tanpa mengorbankan kualitas visual secara signifikan.



Gambar 2.1. Struktur Data Quadtree dalam Kompresi Gambar

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

2.3. Error Measurement Methods

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

2.3.1 Variance

Variansi menghitung seberapa jauh nilai piksel menyimpang dari nilai rata-ratanya untuk setiap kanal warna (R, G, dan B). Rumus yang digunakan adalah:

- Variansi per kanal:

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

Di mana σ_c^2 adalah variansi kanal warna c , $P_{i,c}$ adalah nilai piksel ke- i pada kanal tersebut, μ_c adalah rata-rata nilai piksel dalam blok, dan N adalah jumlah piksel.

- Variansi keseluruhan RGB:

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

Nilai σ_{RGB} inilah yang dibandingkan dengan threshold. Jika lebih besar, maka blok dianggap tidak homogen dan akan dibagi lebih lanjut menjadi sub-blok.

2.3.1 Mean Absolute Deviation (MAD)

MAD menghitung nilai absolut dari selisih antara setiap piksel dan nilai rata-ratanya untuk masing-masing kanal warna (R, G, dan B). Rumus yang digunakan adalah:

- MAD per kanal:

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

Di mana MAD_c adalah MAD kanal warna c , $P_{i,c}$ adalah nilai piksel ke- i pada kanal tersebut, μ_c adalah rata-rata nilai piksel dalam blok, dan N adalah jumlah piksel.

- MAD keseluruhan RGB:

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

Nilai MAD_{RGB} inilah yang dibandingkan dengan threshold. Jika lebih besar, maka blok dianggap tidak homogen dan akan dibagi lebih lanjut menjadi sub-blok.

2.3.1 Max Pixel Difference (MPD)

Max Pixel Difference digunakan untuk mengukur seberapa besar penyimpangan nilai piksel maksimum terhadap nilai rata-rata dalam sebuah blok untuk setiap kanal warna (R, G, dan B). Rumus yang digunakan adalah:

- MPD per kanal:

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

Di mana D_c adalah MPD kanal warna c dan $P_{i,c}$ adalah nilai piksel ke- i pada kanal tersebut.

- MPD keseluruhan RGB:

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

Nilai D_{RGB} inilah yang dibandingkan dengan threshold. Jika lebih besar, maka blok dianggap tidak homogen dan akan dibagi lebih lanjut menjadi sub-blok.

2.3.1 Entropy

Nilai entropy yang tinggi menunjukkan distribusi piksel yang beragam, sedangkan nilai rendah menunjukkan blok yang lebih seragam. Untuk setiap kanal warna (R, G, B), entropy dihitung berdasarkan distribusi probabilitas intensitas piksel. Rumus yang digunakan adalah:

- Entropy per kanal:

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

Di mana H_c adalah entropy kanal warna c dan $P_c(i)$ adalah probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B).

- Entropy keseluruhan RGB:

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

Nilai H_{RGB} inilah yang dibandingkan dengan threshold. Jika lebih besar, maka blok dianggap tidak homogen dan akan dibagi lebih lanjut menjadi sub-blok.

BAB III

ALGORITMA PROGRAM

Algoritma utama dalam program berada dalam sebuah fungsi buildTree, yang menghasilkan QuadNode yaitu bagian dari struktur quad tree. Fungsi buildTree itu sendiri dapat dibagi menjadi beberapa bagian, yaitu: basis rekursi, pembagian blok (divide), pemanggilan rekursif (conquer), penggabungan sub-blok (merge), dan evaluasi homogenitas blok (heuristik). Algoritma ini memanfaatkan cara *divide and conquer* disertai heuristik. Hal ini memungkinkan gambar dibagi menjadi blok-blok kecil secara rekursif, sedangkan heuristik digunakan untuk menentukan kapan proses pembagian dapat dihentikan berdasarkan homogenitas warna suatu blok.

3.1. Basis Rekursi

Basis rekursi dipanggil jika lebar dan tinggi (parameter) bernilai satu. Basis ini memiliki tujuan untuk menghentikan proses rekursif pada blok gambar terkecil, yaitu satu piksel. Pada tahap ini, program akan langsung membaca nilai warna piksel tersebut (komponen R, G, dan B), menyimpannya ke dalam struktur data node, serta menghitung jumlah dan kuadrat dari masing-masing komponen warna. Node tersebut akan menjadi simpul daun (leaf node) dalam struktur pohon quadtree, karena tidak dapat dibagi lebih lanjut.

3.2. Pembagian Blok (Divide)

Pembagian blok dilakukan ketika ukuran blok lebih dari 1 piksel dan belum memenuhi kriteria homogenitas. Pada tahap ini, blok akan dibagi menjadi empat bagian yang ukurannya dihitung berdasarkan setengah dari lebar dan tinggi blok induk. Jika ukuran blok ganjil, maka bagian kiri atau atas akan sedikit lebih besar untuk memastikan seluruh area tercover.

Setelah pembagian dilakukan, fungsi buildTree dipanggil secara rekursif untuk masing-masing sub-blok. Keempat sub-blok ini kemudian akan diproses secara terpisah dengan langkah yang sama (rekursif), hingga mencapai basis rekursi atau memenuhi kriteria homogen.

3.3. Pemanggilan Rekursif (Conquer)

Pemanggilan rekursif terjadi setelah proses pembagian blok dilakukan. Fungsi buildTree akan dipanggil kembali sebanyak empat kali untuk masing-masing sub-blok yang telah dibentuk. Setiap pemanggilan akan memproses blok yang lebih kecil menggunakan langkah yang sama: mulai dari pengecekan basis rekursi, pembacaan nilai piksel, hingga evaluasi homogenitas. Proses ini berlangsung secara berulang (rekursif) hingga setiap bagian dari gambar telah diolah dan struktur pohon quadtree terbentuk secara lengkap.

3.4. Penggabungan Sub-Blok (Merge)

Setelah keempat anak node dari sebuah blok berhasil dibangun, langkah selanjutnya adalah menggabungkan informasi statistik dari seluruh sub-blok ke dalam node induk. Informasi yang digabungkan meliputi jumlah warna merah, hijau, biru, dan kuadrat dari masing-masing warna. Selain itu, jumlah total simpul dan kedalaman maksimum dari anak-anak juga dihitung untuk disimpan dalam node induk. Penggabungan ini memungkinkan evaluasi homogenitas dilakukan pada level yang lebih tinggi, yaitu blok hasil gabungan dari sub-blok.

3.5. Evaluasi Homogenitas Blok (Heuristik)

Setelah penggabungan data dari sub-blok selesai, dilakukan evaluasi terhadap homogenitas blok menggunakan nilai variansi dari masing-masing kanal warna. Variansi dihitung berdasarkan rata-rata dan kuadrat dari nilai warna pada blok. Jika hasil variansi lebih kecil atau sama dengan ambang batas (*threshold*) yang telah ditentukan, atau jumlah piksel dalam blok lebih kecil atau sama dengan nilai minimum blok (*minBlockSize*), maka blok dianggap cukup homogen.

Jika blok homogen, maka seluruh sub-blok dihapus (menjadi leaf node), dan blok tersebut akan diwarnai ulang menggunakan warna rata-rata. Namun, jika tidak memenuhi kriteria, maka blok akan tetap mempertahankan anak-anaknya dan berfungsi sebagai simpul internal dalam struktur pohon quadtree.

BAB IV

IMPLEMENTASI

4.1. Struktur Direktori

```
└── Tucil2_13523142/
    ├── bin/
    │   ├── app/
    │   │   └── App.class
    │   ├── core/
    │   │   └── core classes...
    │   ├── util/
    │   │   └── InputScanner.class
    ├── doc/
    │   └── Tucil2_K3_13523142.pdf
    ├── src/
    │   ├── app/
    │   │   └── App.java          # Main driver
    │   ├── core/
    │   │   ├── CompressEntropy.java
    │   │   ├── CompressMAD.java
    │   │   ├── CompressMPD.java
    │   │   ├── CompressQuadtree.java
    │   │   └── CompressVariance.java
    │   ├── util/
    │   │   └── InputScanner.java  # Input dan validasi
    ├── test/
    │   └── test files...
    ├── build.bat
    ├── build.sh
    ├── clean.bat
    ├── clean.sh
    ├── README.md
    ├── run.bat
    └── run.sh
```

Direktori utama proyek adalah Tucil2_13523142/, yang berisi seluruh komponen penting untuk menjalankan program kompresi gambar berbasis quadtree menggunakan berbagai metode pengukuran error. Di dalamnya, terdapat beberapa subdirektori utama, yaitu src/, bin/, doc/, dan test/, serta beberapa file skrip pendukung seperti build.sh, run.bat, dan README.md.

4.1.1 Direktori src/

Direktori src/ menyimpan seluruh kode sumber (source code) dalam bentuk file .java. Struktur src/ dibagi menjadi tiga paket: app/, core/, dan util/. File App.java yang berada di dalam app/ merupakan titik masuk (entry point) dari program. Semua kelas algoritma kompresi berada di dalam paket core/, yaitu CompressEntropy.java, CompressMAD.java, CompressMPD.java, dan CompressVariance.java, yang masing-masing merepresentasikan metode kompresi yang berbeda. File CompressQuadtree.java berfungsi sebagai kelas induk abstrak (abstract class) yang menyediakan kerangka umum untuk proses kompresi berbasis

quadtree. Sementara itu, util/InputScanner.java menangani masukan dari pengguna sekaligus memvalidasinya.

4.1.2 Direktori bin/

Direktori bin/ berisi hasil kompilasi dari program, yaitu file .class. Struktur direktori bin/ mencerminkan struktur yang sama seperti src/, yakni terdiri dari app/, core/, dan util/. Setiap file .java yang dikompilasi dari src/ akan menghasilkan file .class di lokasi yang sesuai di dalam bin/.

4.1.3 Direktori doc/

Direktori doc/ digunakan untuk menyimpan dokumentasi proyek, termasuk laporan akhir berformat PDF (Tucil2_K3_13523142.pdf). Direktori ini bersifat informatif dan digunakan sebagai pelengkap dari kode program.

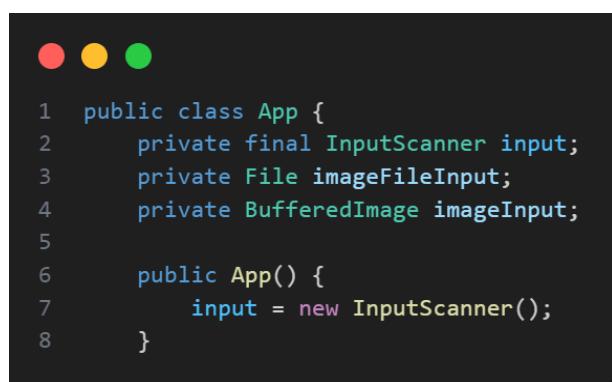
4.1.4 Direktori test/

Direktori test/ ditujukan untuk menyimpan file gambar yang digunakan sebagai bahan uji coba dalam proses kompresi. File-file di dalam direktori ini digunakan untuk menguji akurasi dan efektivitas algoritma kompresi yang telah diimplementasikan.

4.1.5 Direktori pendukung

Selain itu, terdapat juga beberapa file skrip utama seperti build.sh dan build.bat untuk proses kompilasi otomatis, run.sh dan run.bat untuk menjalankan program, serta clean.sh dan clean.bat untuk menghapus file hasil kompilasi. File README.md berisi petunjuk penggunaan program, deskripsi umum, serta panduan input dan output.mogen.

4.2. App Class



```
● ● ●
1 public class App {
2     private final InputScanner input;
3     private File imageFileInput;
4     private BufferedImage imageInput;
5
6     public App() {
7         input = new InputScanner();
8     }
}
```

Gambar 4.2.1. Atribut dan konstruktur App

4.2.1 Method Select Image File

```
● ● ●  
1 public void selectImageFile() {  
2     while (true) {  
3         inputFileInput = input.scanFile("[INPUT] Alamat absolut gambar yang akan dikompresi: ");  
4         try {  
5             imageInput = ImageIO.read(imageFileInput);  
6             if (imageInput != null) {  
7                 System.out.println("Gambar berhasil dimuat: " + imageInput.getWidth() + "x" + imageInput.getHeight());  
8                 break;  
9             } else {  
10                 System.out.println("Gagal membaca gambar. Format mungkin tidak didukung.");  
11             }  
12         } catch (IOException e) {  
13             System.out.println("Terjadi kesalahan saat membaca gambar: " + e.getMessage());  
14         }  
15     }  
16 }
```

Gambar 4.2.2. Fungsi untuk memilih file

4.2.2 Method Print

```
● ● ●  
1 public void printMethods() {  
2     System.out.println("Error Measurement Methods:");  
3     System.out.println("1. Variance");  
4     System.out.println("2. Mean Absolute Deviation (MAD)");  
5     System.out.println("3. Max Pixel Difference");  
6     System.out.println("4. Entropy");  
7 }
```

Gambar 4.2.3. Fungsi menampilkan method

4.2.3 Method Choose Compressor

```
● ● ●  
1 public CompressQuadtree chooseCompressor() {  
2     printMethods();  
3     int method = input.scan("[INPUT] Pilih metode (1-4): ", choice -> {  
4         int val = Integer.parseInt(choice);  
5         if (val < 1 || val > 4) throw new IllegalArgumentException("Harus antara 1 hingga 4.");  
6         return val;  
7     });  
8     return switch (method) {  
9         case 1 -> new CompressVariance(imageFileInput, imageInput, input);  
10        case 2 -> new CompressMAD(imageFileInput, imageInput, input);  
11        case 3 -> new CompressMPD(imageFileInput, imageInput, input);  
12        case 4 -> new CompressEntropy(imageFileInput, imageInput, input);  
13        default -> throw new IllegalStateException("Unexpected method: " + method);  
14    };  
15 }
```

Gambar 4.2.4. Fungsi untuk memilih dan menetapkan tipe compressor

4.2.4 Close and Main Method



```
1 public void close() {
2     input.close();
3 }
4
5 public static void main(String[] args) {
6     App app = new App();
7     app.selectImageFile();
8     CompressQuadtree compressor = app.chooseCompressor();
9     compressor.compress();
10    compressor.saveImageOutput();
11    compressor.output();
12    app.close();
13 }
```

Gambar 4.2.5. Fungsi untuk menutup app dan main driver App

4.3. Input Scanner Class



```
1 public class InputScanner {
2     private final Scanner scanner;
3
4     public InputScanner() {
5         this.scanner = new Scanner(System.in);
6     }
7
8     public <T> T scan(String prompt, Function<String, T> parser) {
9         while (true) {
10             System.out.print(prompt);
11             String input = scanner.nextLine();
12             try {
13                 return parser.apply(input);
14             } catch (Exception e) {
15                 System.out.println("Input tidak valid: " + e.getMessage());
16             }
17         }
18     }
19
20     public File scanFile(String prompt) {
21         return scan(prompt, path -> {
22             File file = new File(path);
23             if (!file.exists()) throw new IllegalArgumentException("File tidak ditemukan.");
24             return file;
25         });
26     }
27
28     public void close() {
29         scanner.close();
30     }
31 }
```

Gambar 4.3.1. Kelas input scanner yang mengintegrasikan validasi input

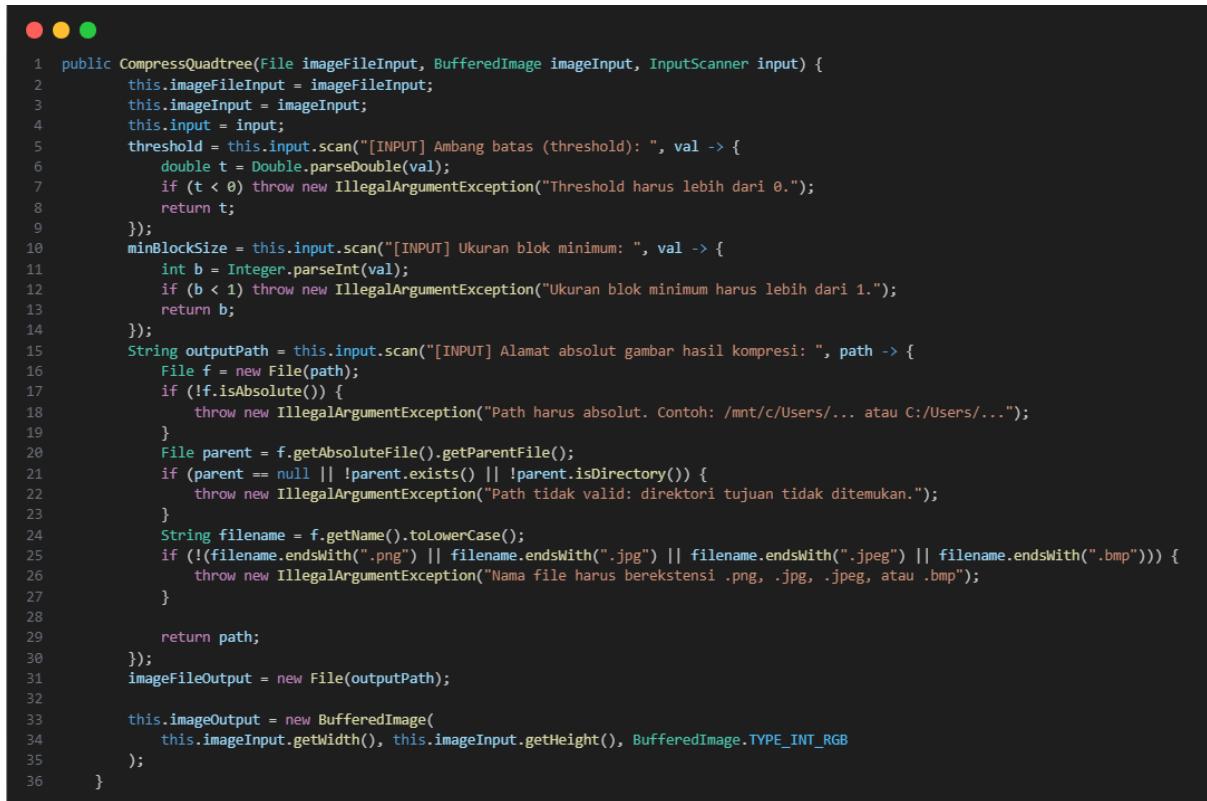
4.4. Compress Quadtree class



```
1 public abstract class CompressQuadtree {
2     protected final InputScanner input;
3     protected File imageFileInput;
4     protected BufferedImage imageInput;
5     protected double threshold;
6     protected int minBlockSize;
7     protected File imageFileOutput;
8
9     protected BufferedImage imageOutput;
10    protected long executionTime;
11    protected int originalSize;
12    protected int compressedSize;
13    protected double actualCompressionRatio;
14    protected int treeDepth;
15    protected int treeNodeCount;
16
17    private long startTime;
```

Gambar 4.4.1. Atribut kelas abstrak compress quadtree

4.4.1 Constructor



```
1 public CompressQuadtree(File imageFileInput, BufferedImage imageInput, InputScanner input) {
2     this.imageFileInput = imageFileInput;
3     this.imageInput = imageInput;
4     this.input = input;
5     threshold = this.input.scan("[INPUT] Ambang batas (threshold): ", val -> {
6         double t = Double.parseDouble(val);
7         if (t < 0) throw new IllegalArgumentException("Threshold harus lebih dari 0.");
8         return t;
9     });
10    minBlockSize = this.input.scan("[INPUT] Ukuran blok minimum: ", val -> {
11        int b = Integer.parseInt(val);
12        if (b < 1) throw new IllegalArgumentException("Ukuran blok minimum harus lebih dari 1.");
13        return b;
14    });
15    String outputPath = this.input.scan("[INPUT] Alamat absolut gambar hasil kompresi: ", path -> {
16        File f = new File(path);
17        if (!f.isAbsolute()) {
18            throw new IllegalArgumentException("Path harus absolut. Contoh: /mnt/c/Users/... atau C:/Users/...");
19        }
20        File parent = f.getAbsoluteFile().getParentFile();
21        if (parent == null || !parent.exists() || !parent.isDirectory()) {
22            throw new IllegalArgumentException("Path tidak valid: direktori tujuan tidak ditemukan.");
23        }
24        String filename = f.getName().toLowerCase();
25        if (!(filename.endsWith(".png") || filename.endsWith(".jpg") || filename.endsWith(".jpeg") || filename.endsWith(".bmp"))) {
26            throw new IllegalArgumentException("Nama file harus berekstensi .png, .jpg, .jpeg, atau .bmp");
27        }
28
29        return path;
30    });
31    imageFileOutput = new File(outputPath);
32
33    this.imageOutput = new BufferedImage(
34        this.imageInput.getWidth(), this.imageInput.getHeight(), BufferedImage.TYPE_INT_RGB
35    );
36 }
```

Gambar 4.4.2. Konstruktor kelas abstrak compress quadtree

4.4.2 Helper Methods

```
● ● ●
1 public abstract void compress();
2
3     protected void logStart() {
4         System.out.println("Starting compression for: " + imageFileInput.getName());
5         startTime = System.nanoTime();
6     }
7
8     protected void logFinish() {
9         long endTime = System.nanoTime();
10        executionTime = endTime - startTime;
11    }
12
13    private void updateFileSizes() {
14        originalSize = (int) imageFileInput.length();
15        File outputFile = new File(imageFileOutput.getAbsolutePath());
16        if (outputFile.exists()) {
17            compressedSize = (int) outputFile.length();
18        }
19    }
20
21    public void saveImageOutput() {
22        try {
23            String formatName = "png";
24            ImageIO.write(imageOutput, formatName, imageFileOutput);
25            updateFileSizes();
26        } catch (IOException e) {
27            System.out.println("Gagal menyimpan gambar: " + e.getMessage());
28        }
29    }
30
31    public void output() {
32        System.out.printf("[OUTPUT] Waktu eksekusi      : %.2f ms\n", executionTime / 1_000_000.0);
33        System.out.printf("[OUTPUT] Ukuran gambar sebelum   : %.2F KB (%d bytes)\n", originalSize / 1024.0, originalSize);
34        System.out.printf("[OUTPUT] Ukuran gambar setelah   : %.2F KB (%d bytes)\n", compressedSize / 1024.0, compressedSize);
35        if (originalSize > 0) {
36            actualCompressionRatio = (1 - ((double) compressedSize / originalSize)) * 100;
37            System.out.printf("[OUTPUT] Persentase kompresi    : %.2f%%\n", actualCompressionRatio);
38        } else {
39            System.out.println("[OUTPUT] Persentase kompresi    : N/A (ukuran asli 0)");
40        }
41        System.out.printf("[OUTPUT] Kedalaman pohon       : %d\n", treeDepth);
42        System.out.printf("[OUTPUT] Banyak simpul        : %d\n", treeNodeCount);
43        System.out.printf("[OUTPUT] Gambar output disimpan di: %s\n", imageFileOutput.getAbsolutePath());
44    }
45 }
```

Gambar 4.4.3. Metode pendukung dalam kelas abstrak compress quadtree

4.4.3 Abstract Quad Node and Quad Builder Class

```
● ● ●
1 protected abstract static class QuadNode {
2     public int depth = 1;
3     public int totalNodes = 1;
4     public QuadNode[] children = new QuadNode[4];
5
6     public QuadNode() {
7     }
8 }
9
10 protected abstract class QuadBuilder {
11     protected double threshold;
12     protected int minBlockSize;
13
14     public QuadBuilder(double threshold, int minBlockSize) {
15         this.threshold = threshold;
16         this.minBlockSize = minBlockSize;
17     }
18
19     public abstract QuadNode buildTree(BufferedImage image, BufferedImage result, int x, int y, int width, int height);
20 }
```

Gambar 4.4.4. Struktur dan atribut utama dari kelas abstrak QuadNode dan QuadBuilder

4.5. Compress Variance Class

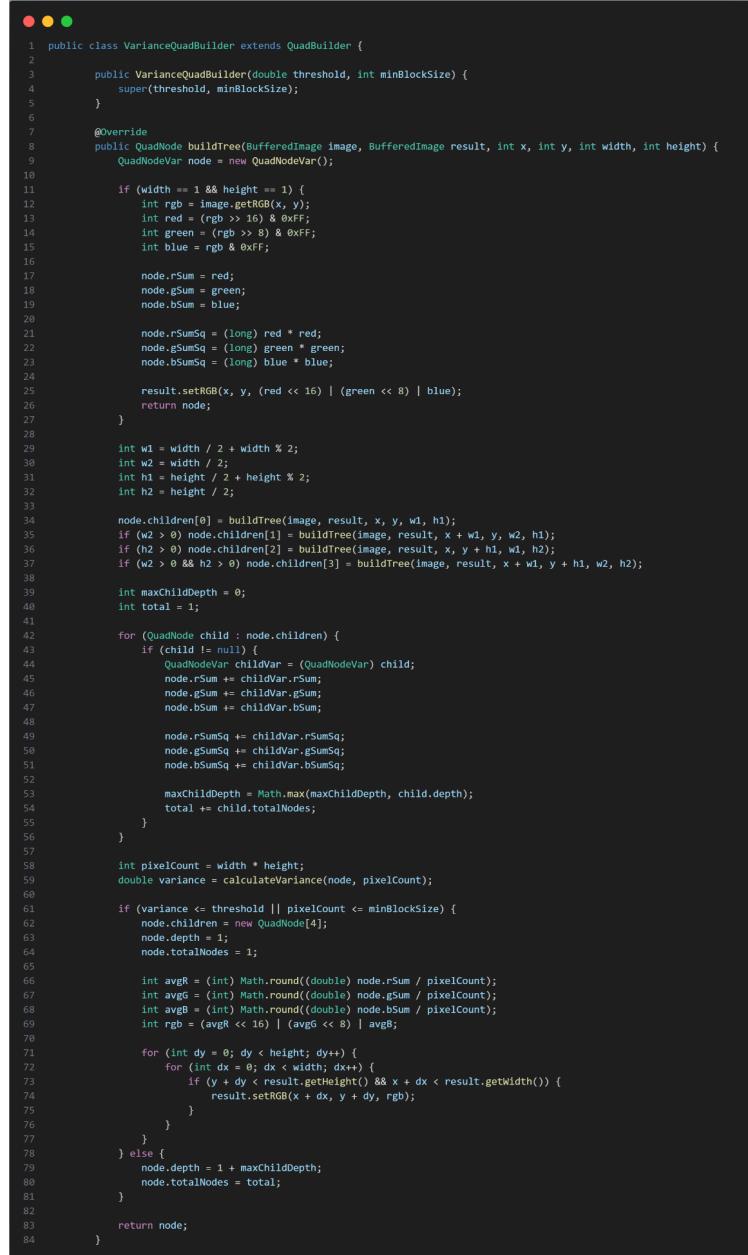
```
 1 public class CompressVariance extends CompressQuadtree {
 2     public CompressVariance(File imageFileInput, BufferedImage imageInput, InputScanner input) {
 3         super(imageFileInput, imageInput, input);
 4     }
 5
 6     @Override
 7     public void compress(){
 8         logStart();
 9         QuadBuilder builder = new VarianceQuadBuilder(this.threshold, this.minBlockSize);
10         QuadNode tree = builder.buildTree(
11             this.imageInput, imageOutput, 0, 0,
12             this.imageInput.getWidth(), this.imageInput.getHeight()
13         );
14         this.treeNodeCount = tree.totalNodes;
15         this.treeDepth = tree.depth;
16         logFinish();
17     }
18
19     @Override
20     public void output(){
21         System.out.println();
22         System.out.println("===== HASIL KOMPRESI METODE VARIANCE =====");
23         super.output();
24     }
}
```

Gambar 4.5.1. Implementasi metode compress() dan output() dalam kelas CompressVariance
4.5.1 Quad Node Variance Class

```
 1 private static class QuadNodeVar extends QuadNode {
 2     public long rSum, gSum, bSum;
 3     public long rSumSq, gSumSq, bSumSq;
 4     public QuadNodeVar() {
 5         super();
 6     }
 7 }
```

Gambar 4.5.2. Implementasi kelas QuadNodeVar sebagai turunan dari QuadNode, yang menyimpan total dan kuadrat jumlah warna untuk perhitungan variansi pada kompresi Quadtree metode Variance.

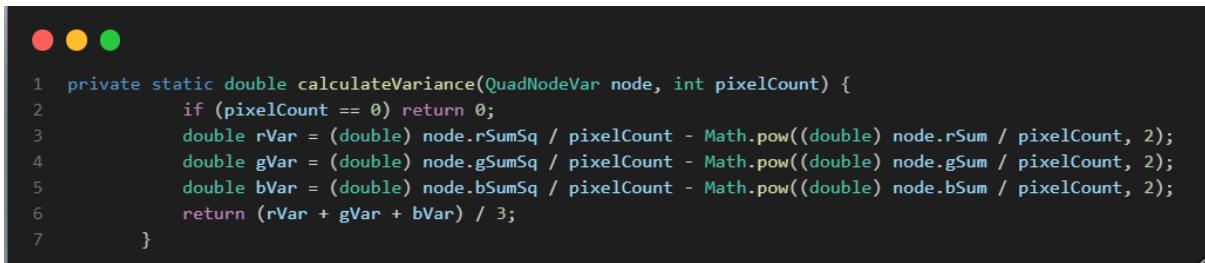
4.5.2 Quad Builder Variance Class



```
1 public class VarianceQuadBuilder extends QuadBuilder {
2
3     public VarianceQuadBuilder(double threshold, int minBlockSize) {
4         super(threshold, minBlockSize);
5     }
6
7     @Override
8     public QuadNode buildTree(BufferedImage image, BufferedImage result, int x, int y, int width, int height) {
9         QuadNodeVar node = new QuadNodeVar();
10
11        if (width == 1 && height == 1) {
12            int rgb = image.getRGB(x, y);
13            int red = (rgb >> 16) & 0xFF;
14            int green = (rgb >> 8) & 0xFF;
15            int blue = rgb & 0xFF;
16
17            node.rSum = red;
18            node.gSum = green;
19            node.bSum = blue;
20
21            node.rSumSq = (long) red * red;
22            node.gSumSq = (long) green * green;
23            node.bSumSq = (long) blue * blue;
24
25            result.setRGB(x, y, (red << 16) | (green << 8) | blue);
26            return node;
27        }
28
29        int w1 = width / 2 + width % 2;
30        int w2 = width / 2;
31        int h1 = height / 2 + height % 2;
32        int h2 = height / 2;
33
34        node.children[0] = buildTree(image, result, x, y, w1, h1);
35        if (w2 > 0) node.children[1] = buildTree(image, result, x + w1, y, w2, h1);
36        if (h2 > 0) node.children[2] = buildTree(image, result, x, y + h1, w1, h2);
37        if (w2 > 0 && h2 > 0) node.children[3] = buildTree(image, result, x + w1, y + h1, w2, h2);
38
39        int maxChildDepth = 0;
40        int total = 1;
41
42        for (QuadNode child : node.children) {
43            if (child != null) {
44                QuadNodeVar childVar = (QuadNodeVar) child;
45                node.rSum += childVar.rSum;
46                node.gSum += childVar.gSum;
47                node.bSum += childVar.bSum;
48
49                node.rSumSq += childVar.rSumSq;
50                node.gSumSq += childVar.gSumSq;
51                node.bSumSq += childVar.bSumSq;
52
53                maxChildDepth = Math.max(maxChildDepth, child.depth);
54                total += child.totalNodes;
55            }
56        }
57
58        int pixelCount = width * height;
59        double variance = calculateVariance(node, pixelCount);
60
61        if (variance <= threshold || pixelCount <= minBlockSize) {
62            node.children = new QuadNode[4];
63            node.depth = 1;
64            node.totalNodes = 1;
65
66            int avgR = (int) Math.round((double) node.rSum / pixelCount);
67            int avgG = (int) Math.round((double) node.gSum / pixelCount);
68            int avgB = (int) Math.round((double) node.bSum / pixelCount);
69            int rgb = (avgR << 16) | (avgG << 8) | avgB;
70
71            for (int dy = 0; dy < height; dy++) {
72                for (int dx = 0; dx < width; dx++) {
73                    if (y + dy < result.getHeight() && x + dx < result.getWidth()) {
74                        result.setRGB(x + dx, y + dy, rgb);
75                    }
76                }
77            }
78        } else {
79            node.depth = 1 + maxChildDepth;
80            node.totalNodes = total;
81        }
82
83        return node;
84    }
}
```

Gambar 4.5.3. Implementasi lengkap metode buildTree() dalam kelas VarianceQuadBuilder yang menerapkan algoritma divide and conquer serta evaluasi homogenitas blok berdasarkan variansi.

4.5.3 Calculate Variance



```
1 private static double calculateVariance(QuadNodeVar node, int pixelCount) {
2     if (pixelCount == 0) return 0;
3     double rVar = (double) node.rSumSq / pixelCount - Math.pow((double) node.rSum / pixelCount, 2);
4     double gVar = (double) node.gSumSq / pixelCount - Math.pow((double) node.gSum / pixelCount, 2);
5     double bVar = (double) node.bSumSq / pixelCount - Math.pow((double) node.bSum / pixelCount, 2);
6     return (rVar + gVar + bVar) / 3;
7 }
```

Gambar 4.5.3. Fungsi pendukung untuk menghitung variansi

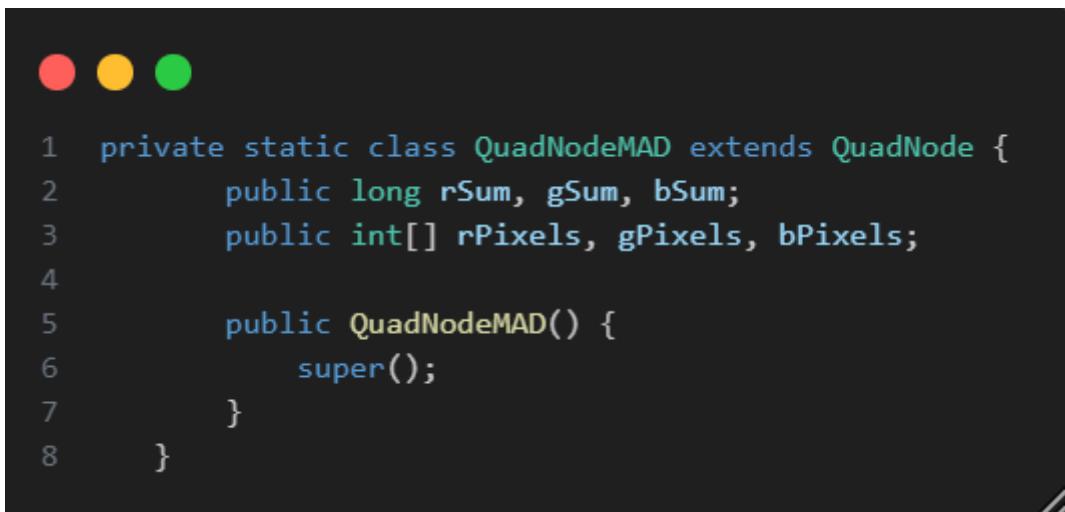
4.6. Compress MAD Class



```
1 public class CompressMAD extends CompressQuadtree {
2     public CompressMAD(File inputFile, BufferedImage imageInput, InputScanner input) {
3         super(inputFile, imageInput, input);
4     }
5
6     @Override
7     public void compress() {
8         logStart();
9         QuadBuilder builder = new MADQuadBuilder(this.threshold, this.minBlockSize);
10        QuadNode tree = builder.buildTree(
11            this.imageInput, imageOutput, 0, 0,
12            this.imageInput.getWidth(), this.imageInput.getHeight()
13        );
14        this.treeNodeCount = tree.totalNodes;
15        this.treeDepth = tree.depth;
16        logFinish();
17    }
18
19    @Override
20    public void output() {
21        System.out.println();
22        System.out.println("===== HASIL KOMPRESI METODE MAD =====");
23        super.output();
24    }
}
```

Gambar 4.6.1. Implementasi metode compress() dan output() dalam kelas CompressMAD

4.6.1 Quad Node MAD Class



```
1 private static class QuadNodeMAD extends QuadNode {
2     public long rSum, gSum, bSum;
3     public int[] rPixels, gPixels, bPixels;
4
5     public QuadNodeMAD() {
6         super();
7     }
8 }
```

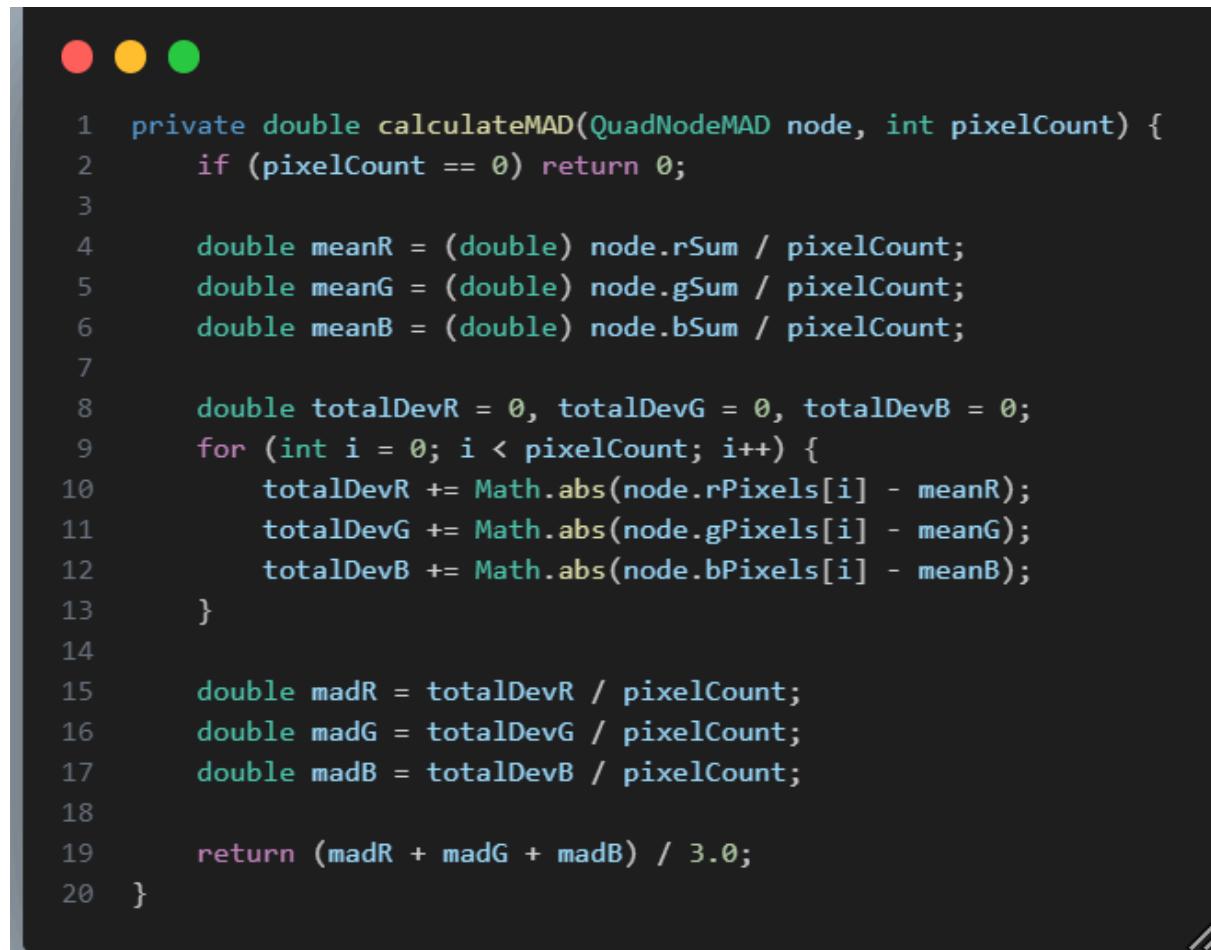
Gambar 4.6.2. Implementasi kelas QuadNodeMAD sebagai turunan dari QuadNode, yang menyimpan total warna dan pixel array untuk perhitungan MAD pada kompresi Quadtree metode MAD.

4.6.2 Quad Builder MAD Class

```
 1  private class MADQuadBuilder extends QuadBuilder {
 2      public MADQuadBuilder(double threshold, int minBlockSize) {
 3          super(threshold, minBlockSize);
 4      }
 5
 6      @Override
 7      public QuadNode buildTree(BufferedImage image, BufferedImage result, int x, int y, int width, int height) {
 8          QuadNodeMAD node = new QuadNodeMAD();
 9          int pixelCount = width * height;
10
11          if (width == 1 && height == 1) {
12              int rgb = image.getRGB(x, y);
13              int red = (rgb >> 16) & 0xFF;
14              int green = (rgb >> 8) & 0xFF;
15              int blue = (rgb & 0xFF);
16
17              node.rSum = red;
18              node.gSum = green;
19              node.bSum = blue;
20
21              node.rPixels = new int[]{red};
22              node.gPixels = new int[]{green};
23              node.bPixels = new int[]{blue};
24
25              result.setRGB(x, y, (red << 16) | (green << 8) | blue);
26              return node;
27          }
28
29          int w1 = width / 2 + width % 2;
30          int w2 = width / 2;
31          int h1 = height / 2 + height % 2;
32          int h2 = height / 2;
33
34          node.children[0] = buildTree(image, result, x, y, w1, h1);
35          if (w2 > 0) node.children[1] = buildTree(image, result, x + w1, y, w2, h1);
36          if (h2 > 0) node.children[2] = buildTree(image, result, x, y + h1, w1, h2);
37          if (w2 > 0 && h2 > 0) node.children[3] = buildTree(image, result, x + w1, y + h1, w2, h2);
38
39          node.rPixels = new int[pixelCount];
40          node.gPixels = new int[pixelCount];
41          node.bPixels = new int[pixelCount];
42
43          int i = 0;
44          int maxDepth = 0;
45          int totalNodes = 1;
46
47          for (QuadNode child : node.children) {
48              if (child != null) {
49                  QuadNodeMAD childNode = (QuadNodeMAD) child;
50
51                  node.rSum += childNode.rSum;
52                  node.gSum += childNode.gSum;
53                  node.bSum += childNode.bSum;
54
55                  System.arraycopy(childNode.rPixels, 0, node.rPixels, i, childNode.rPixels.length);
56                  System.arraycopy(childNode.gPixels, 0, node.gPixels, i, childNode.gPixels.length);
57                  System.arraycopy(childNode.bPixels, 0, node.bPixels, i, childNode.bPixels.length);
58                  i += childNode.rPixels.length;
59
60                  maxDepth = Math.max(maxDepth, child.depth);
61                  totalNodes += child.totalNodes;
62              }
63          }
64
65          double mad = calculateMAD(node, pixelCount);
66
67          if (mad <= threshold || pixelCount <= minBlockSize) {
68              node.children = new QuadNode[4];
69              node.depth = 1;
70              node.totalNodes = 1;
71
72              int avgR = (int) Math.round(((double) node.rSum / pixelCount));
73              int avgG = (int) Math.round(((double) node.gSum / pixelCount));
74              int avgB = (int) Math.round(((double) node.bSum / pixelCount));
75              int rgb = (avgR << 16) | (avgG << 8) | avgB;
76
77              for (int dy = 0; dy < height; dy++) {
78                  for (int dx = 0; dx < width; dx++) {
79                      if (y + dy < result.getHeight() && x + dx < result.getWidth()) {
80                          result.setRGB(x + dx, y + dy, rgb);
81                      }
82                  }
83              }
84          } else {
85              node.depth = 1 + maxDepth;
86              node.totalNodes = totalNodes;
87          }
88
89      }
90  }
```

Gambar 4.6.3. Implementasi lengkap metode buildTree() dalam kelas QuadBuilderMAD yang menerapkan algoritma divide and conquer serta evaluasi homogenitas blok berdasarkan MAD.

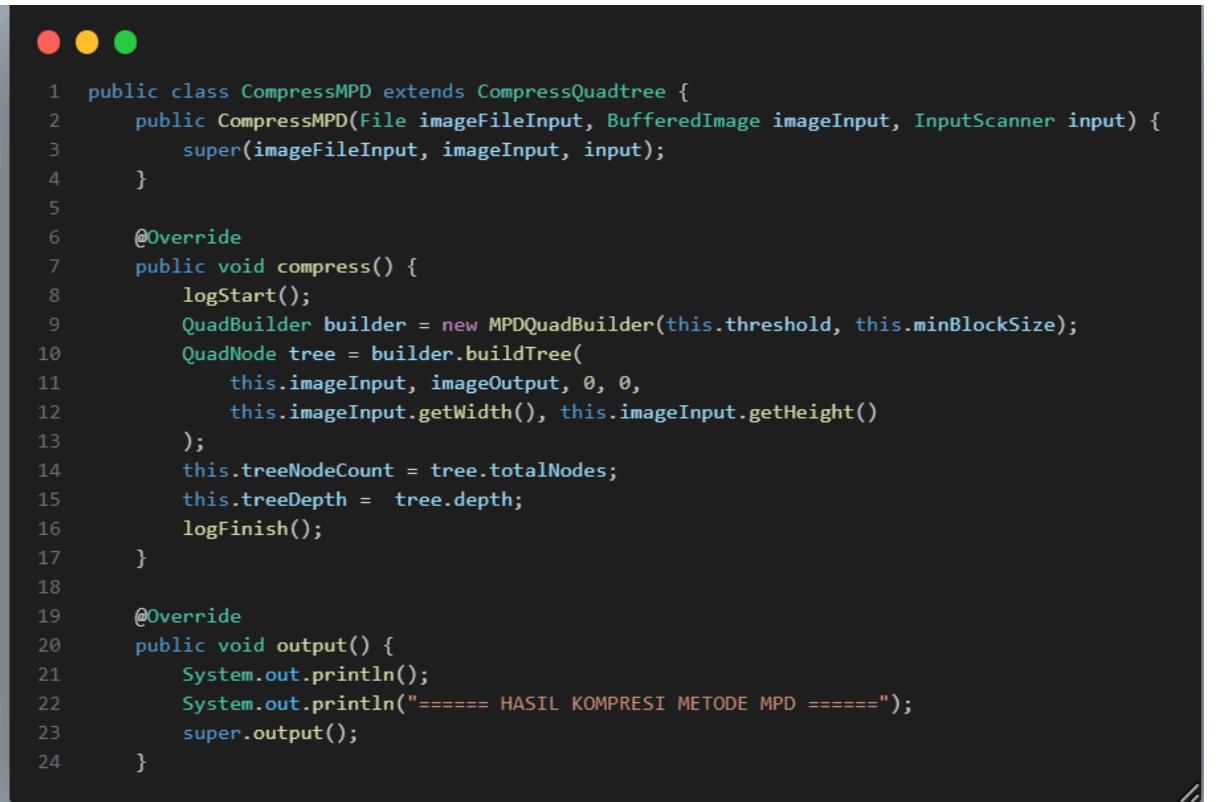
4.6.3 Calculate MAD



```
1  private double calculateMAD(QuadNodeMAD node, int pixelCount) {  
2      if (pixelCount == 0) return 0;  
3  
4      double meanR = (double) node.rSum / pixelCount;  
5      double meanG = (double) node.gSum / pixelCount;  
6      double meanB = (double) node.bSum / pixelCount;  
7  
8      double totalDevR = 0, totalDevG = 0, totalDevB = 0;  
9      for (int i = 0; i < pixelCount; i++) {  
10          totalDevR += Math.abs(node.rPixels[i] - meanR);  
11          totalDevG += Math.abs(node.gPixels[i] - meanG);  
12          totalDevB += Math.abs(node.bPixels[i] - meanB);  
13      }  
14  
15      double madR = totalDevR / pixelCount;  
16      double madG = totalDevG / pixelCount;  
17      double madB = totalDevB / pixelCount;  
18  
19      return (madR + madG + madB) / 3.0;  
20  }
```

Gambar 4.6.3. Fungsi pendukung untuk menghitung MAD

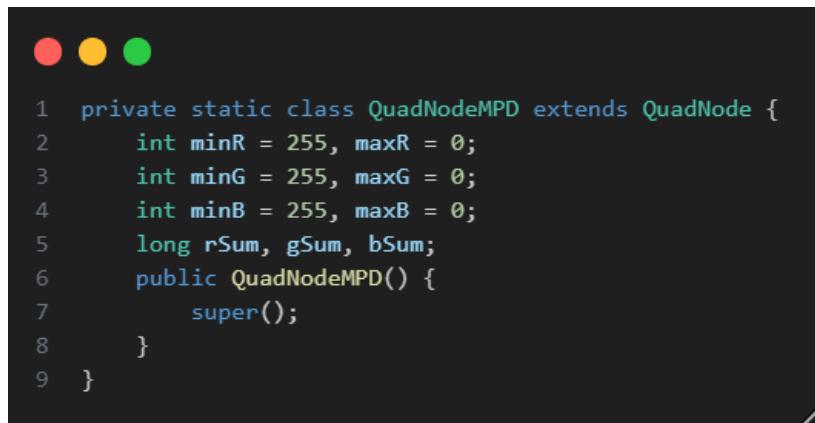
4.7. Compress MPD class



```
1 public class CompressMPD extends CompressQuadtree {
2     public CompressMPD(File imageFileInput, BufferedImage imageInput, InputScanner input) {
3         super(imageFileInput, imageInput, input);
4     }
5
6     @Override
7     public void compress() {
8         logStart();
9         QuadBuilder builder = new MPDQuadBuilder(this.threshold, this.minBlockSize);
10        QuadNode tree = builder.buildTree(
11            this.imageInput, imageOutput, 0, 0,
12            this.imageInput.getWidth(), this.imageInput.getHeight()
13        );
14        this.treeNodeCount = tree.totalNodes;
15        this.treeDepth = tree.depth;
16        logFinish();
17    }
18
19    @Override
20    public void output() {
21        System.out.println();
22        System.out.println("===== HASIL KOMPRESI METODE MPD =====");
23        super.output();
24    }

```

4.7.1 Quad Node MPD Class



```
1 private static class QuadNodeMPD extends QuadNode {
2     int minR = 255, maxR = 0;
3     int minG = 255, maxG = 0;
4     int minB = 255, maxB = 0;
5     long rSum, gSum, bSum;
6     public QuadNodeMPD() {
7         super();
8     }
9 }
```

Gambar 4.7.2. Implementasi kelas QuadNodeMPD sebagai turunan dari QuadNode, yang menyimpan nilai maksimum dan minimum untuk perhitungan MPD pada kompresi Quadtree metode MPD.

4.7.2 Quad Builder MPD Class

```
● ● ●
1  private class MPDQuadBuilder extends QuadBuilder {
2      public MPDQuadBuilder(double threshold, int minBlockSize) {
3          super(threshold, minBlockSize);
4      }
5
6      @Override
7      public QuadNode buildTree(BufferedImage image, BufferedImage result, int x, int y, int width, int height) {
8          QuadNodeMPD node = new QuadNodeMPD();
9
10         if (width == 1 && height == 1) {
11             int rgb = image.getRGB(x, y);
12             int r = (rgb >> 16) & 0xFF;
13             int g = (rgb >> 8) & 0xFF;
14             int b = rgb & 0xFF;
15
16             node.rSum = r;
17             node.gSum = g;
18             node.bSum = b;
19
20             node.minR = node.maxR = r;
21             node.minG = node.maxG = g;
22             node.minB = node.maxB = b;
23
24             result.setRGB(x, y, rgb);
25             return node;
26         }
27
28         int w1 = width / 2 + width % 2;
29         int w2 = width / 2;
30         int h1 = height / 2 + height % 2;
31         int h2 = height / 2;
32
33         node.children[0] = buildTree(image, result, x, y, w1, h1);
34         if (w2 > 0) node.children[1] = buildTree(image, result, x + w1, y, w2, h1);
35         if (h2 > 0) node.children[2] = buildTree(image, result, x, y + h1, w1, h2);
36         if (w2 > 0 && h2 > 0) node.children[3] = buildTree(image, result, x + w1, y + h1, w2, h2);
37
38         int maxChildDepth = 0;
39         int total = 1;
40
41         for (QuadNode child : node.children) {
42             if (child != null) {
43                 QuadNodeMPD childNode = (QuadNodeMPD) child;
44
45                 node.rSum += childNode.rSum;
46                 node.gSum += childNode.gSum;
47                 node.bSum += childNode.bSum;
48
49                 node.minR = Math.min(node.minR, childNode.minR);
50                 node.maxR = Math.max(node.maxR, childNode.maxR);
51                 node.minG = Math.min(node.minG, childNode.minG);
52                 node.maxG = Math.max(node.maxG, childNode.maxG);
53                 node.minB = Math.min(node.minB, childNode.minB);
54                 node.maxB = Math.max(node.maxB, childNode.maxB);
55
56                 maxChildDepth = Math.max(maxChildDepth, child.depth);
57                 total += child.totalNodes;
58             }
59         }
60
61         int pixelCount = width * height;
62         double diff = calculateMaxDiff(node);
63
64         if (diff <= threshold || pixelCount <= minBlockSize) {
65             node.children = new QuadNode[4];
66             node.depth = 1;
67             node.totalNodes = 1;
68
69             int avgR = (int) Math.round((double) node.rSum / pixelCount);
70             int avgG = (int) Math.round((double) node.gSum / pixelCount);
71             int avgB = (int) Math.round((double) node.bSum / pixelCount);
72             int rgb = (avgR << 16) | (avgG << 8) | avgB;
73
74             for (int dy = 0; dy < height; dy++) {
75                 for (int dx = 0; dx < width; dx++) {
76                     if (y + dy < result.getHeight() && x + dx < result.getWidth()) {
77                         result.setRGB(x + dx, y + dy, rgb);
78                     }
79                 }
80             }
81         } else {
82             node.depth = 1 + maxChildDepth;
83             node.totalNodes = total;
84         }
85
86         return node;
87     }
}
```

Gambar 4.7.3. Implementasi lengkap metode buildTree() dalam kelas QuadBuilderMPD yang menerapkan algoritma divide and conquer serta evaluasi homogenitas blok berdasarkan MPD

4.7.3 Calculate Max Difference



```
1 private double calculateMaxDiff(QuadNodeMPD node) {
2     int dR = node.maxR - node.minR;
3     int dG = node.maxG - node.minG;
4     int dB = node.maxB - node.minB;
5     return (dR + dG + dB) / 3.0;
6 }
```

Gambar 4.6.3. Fungsi pendukung untuk menghitung MPD

4.8. Compress Entropy Class



```
1 public class CompressEntropy extends CompressQuadtree {
2     public CompressEntropy(File imageFileInput, BufferedImage imageInput, InputScanner input) {
3         super(imageFileInput, imageInput, input);
4     }
5
6     @Override
7     public void compress() {
8         logStart();
9         QuadBuilder builder = new EntropyQuadBuilder(this.threshold, this.minBlockSize);
10        QuadNode tree = builder.buildTree(
11            this.imageInput, imageOutput, 0, 0,
12            this.imageInput.getWidth(), this.imageInput.getHeight()
13        );
14        this.treeNodeCount = tree.totalNodes;
15        this.treeDepth = tree.depth;
16        logFinish();
17    }
18
19    @Override
20    public void output(){
21        System.out.println();
22        System.out.println("===== HASIL KOMPRESI METODE ENTROPY =====");
23        super.output();
24    }
}
```

4.8.1 Quad Node Entropy Class



```
1  private static class QuadNodeEntropy extends QuadNode {  
2      public long rSum, gSum, bSum;  
3      public long rSumSq, gSumSq, bSumSq;  
4      public int[] rListCount = new int[256];  
5      public int[] gListCount = new int[256];  
6      public int[] bListCount = new int[256];  
7  
8      public QuadNodeEntropy() {  
9          super();  
10     }  
11 }
```

Gambar 4.8.2. Implementasi kelas QuadNodeMPD sebagai turunan dari QuadNode, yang menyimpan total dan kuadrat jumlah warna, serta array distribusi pixel untuk perhitungan entropy pada kompresi Quadtree metode entropy.

4.8.2 Quad Builder Entropy Class

```
● ● ●
1  private class EntropyQuadBuilder extends QuadBuilder {
2      public EntropyQuadBuilder(double threshold, int minBlockSize) {
3          super(threshold, minBlockSize);
4      }
5
6      @Override
7      public QuadNode buildTree(BufferedImage image, BufferedImage result, int x, int y, int width, int height) {
8          QuadNodeEntropy node = new QuadNodeEntropy();
9
10         if (width == 1 && height == 1) {
11             int rgb = image.getRGB(x, y);
12             int red = (rgb >> 16) & 0xFF;
13             int green = (rgb >> 8) & 0xFF;
14             int blue = rgb & 0xFF;
15
16             node.rSum = red;
17             node.gSum = green;
18             node.bSum = blue;
19
20             node.rSumSq = (long) red * red;
21             node.gSumSq = (long) green * green;
22             node.bSumSq = (long) blue * blue;
23
24             node.rListCount[red]++;
25             node.gListCount[green]++;
26             node.bListCount[blue]++;
27
28             result.setRGB(x, y, rgb);
29             return node;
30         }
31
32         int w1 = width / 2 + width % 2;
33         int w2 = width / 2;
34         int h1 = height / 2 + height % 2;
35         int h2 = height / 2;
36
37         node.children[0] = buildTree(image, result, x, y, w1, h1);
38         if (w2 > 0) node.children[1] = buildTree(image, result, x + w1, y, w2, h1);
39         if (h2 > 0) node.children[2] = buildTree(image, result, x, y + h1, w1, h2);
40         if (w2 > 0 && h2 > 0) node.children[3] = buildTree(image, result, x + w1, y + h1, w2, h2);
41
42         int maxChildDepth = 0;
43         int total = 1;
44
45         for (QuadNode child : node.children) {
46             if (child != null) {
47                 QuadNodeEntropy childNode = (QuadNodeEntropy) child;
48
49                 node.rSum += childNode.rSum;
50                 node.gSum += childNode.gSum;
51                 node.bSum += childNode.bSum;
52
53                 node.rSumSq += childNode.rSumSq;
54                 node.gSumSq += childNode.gSumSq;
55                 node.bSumSq += childNode.bSumSq;
56
57                 for (int i = 0; i < 256; i++) {
58                     node.rListCount[i] += childNode.rListCount[i];
59                     node.gListCount[i] += childNode.gListCount[i];
60                     node.bListCount[i] += childNode.bListCount[i];
61                 }
62
63                 maxChildDepth = Math.max(maxChildDepth, child.depth);
64                 total += child.totalNodes;
65             }
66         }
67
68         int pixelCount = width * height;
69         double entropy = calculateEntropy(node, pixelCount);
70
71         if (entropy <= threshold || pixelCount <= minBlockSize) {
72             node.children = new QuadNode[4];
73             node.depth = 1;
74             node.totalNodes = 1;
75
76             int avgR = (int) Math.round((double) node.rSum / pixelCount);
77             int avgG = (int) Math.round((double) node.gSum / pixelCount);
78             int avgB = (int) Math.round((double) node.bSum / pixelCount);
79             int rgb = (avgR << 16) | (avgG << 8) | avgB;
80
81             for (int dy = 0; dy < height; dy++) {
82                 for (int dx = 0; dx < width; dx++) {
83                     if (y + dy < result.getHeight() && x + dx < result.getWidth()) {
84                         result.setRGB(x + dx, y + dy, rgb);
85                     }
86                 }
87             }
88         } else {
89             node.depth = 1 + maxChildDepth;
90             node.totalNodes = total;
91         }
92
93         return node;
94     }
}
```

Gambar 4.8.3. Implementasi lengkap metode buildTree() dalam kelas QuadBuilderEntropy yang menerapkan algoritma divide and conquer serta evaluasi homogenitas blok berdasarkan Entropy

4.8.3 Calculate Entropy Class



```
1 private double calculateEntropy(QuadNodeEntropy node, int pixelCount) {
2     if (pixelCount == 0) return 0;
3
4     double rEntropy = entropyFromHistogram(node.rListCount, pixelCount);
5     double gEntropy = entropyFromHistogram(node.gListCount, pixelCount);
6     double bEntropy = entropyFromHistogram(node.bListCount, pixelCount);
7
8     return (rEntropy + gEntropy + bEntropy) / 3.0;
9 }
10
11 private double entropyFromHistogram(int[] histogram, int totalPixels) {
12     double entropy = 0;
13     for (int count : histogram) {
14         if (count == 0) continue;
15         double p = (double) count / totalPixels;
16         entropy += p * (Math.log(p) / Math.log(2));
17     }
18     return -entropy;
19 }
```

Gambar 4.8.3. Fungsi pendukung untuk menghitung entropy

BAB V

EKSPERIMENT DAN ANALISIS

5.1. Pengujian Program

Pengujian dilakukan sebanyak delapan (8) kali, dengan setiap metode yang diimplementasi diuji sebanyak dua kali dengan gambar yang berbeda-beda.

5.1.1 Pengujian 1

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase1.png Gambar berhasil dimuat: 3840x2160 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 1 [INPUT] Ambang batas (threshold): 150 [INPUT] Ukuran blok minimum: 1 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output1.png Input tidak valid: Path harus absolut. Contoh: /mnt/c/Users/... atau C:/Users/... [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output1.png Starting compression for: testcase1.png</pre>	<pre>===== HASIL KOMPRESI METODE VARIANCE ===== [OUTPUT] Waktu eksekusi : 1669,92 ms [OUTPUT] Ukuran gambar sebelum : 5217,32 KB (5342534 bytes) [OUTPUT] Ukuran gambar setelah : 426,46 KB (436698 bytes) [OUTPUT] Persentase kompresi : 91,83% [OUTPUT] Kedalaman pohon : 13 [OUTPUT] Banyak simpul : 58211 [OUTPUT] Gambar output disimpan di: C:\test\output1.png</pre>
Gambar		

Tabel 5.1.1. Pengujian menggunakan metode variansi (1)

5.1.2 Pengujian 2

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase2.png Gambar berhasil dimuat: 4640x2680 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 1 [INPUT] Ambang batas (threshold): 300 [INPUT] Ukuran blok minimum: 1 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output2.png Starting compression for: testcase2.png</pre>	<pre>===== HASIL KOMPRESI METODE VARIANCE ===== [OUTPUT] Waktu eksekusi : 2618,96 ms [OUTPUT] Ukuran gambar sebelum : 6229,35 KB (637885 bytes) [OUTPUT] Ukuran gambar setelah : 514,31 KB (526654 bytes) [OUTPUT] Persentase kompresi : 91,74% [OUTPUT] Kedalaman pohon : 14 [OUTPUT] Banyak simpul : 151275 [OUTPUT] Gambar output disimpan di: C:\test\output2.png</pre>
Gambar		

Tabel 5.1.2. Pengujian menggunakan metode variansi (2)

5.1.3 Pengujian 3

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase3.jpg Gambar berhasil dimuat: 6174x3472 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 2 [INPUT] Ambang batas (threshold): 10 [INPUT] Ukuran blok minimum: 4 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output3.jpg Starting compression for: testcase3.jpg</pre>	<pre>===== HASIL KOMPRESI METODE MAD ===== [OUTPUT] Waktu eksekusi : 8564,88 ms [OUTPUT] Ukuran gambar sebelum : 3912,33 KB (4006223 bytes) [OUTPUT] Ukuran gambar setelah : 3586,16 KB (3672228 bytes) [OUTPUT] Persentase kompresi : 8,34% [OUTPUT] Kedalaman pohon : 13 [OUTPUT] Banyak simpul : 867145 [OUTPUT] Gambar output disimpan di: C:/test/output3.jpg</pre>
Gambar		

Tabel 5.1.3. Pengujian menggunakan metode MAD (1)

5.1.4 Pengujian 4

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase4.jpg Gambar berhasil dimuat: 4500x3048 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 2 [INPUT] Ambang batas (threshold): 20 [INPUT] Ukuran blok minimum: 4 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output4.jpg Starting compression for: testcase4.jpg</pre>	<pre>===== HASIL KOMPRESI METODE MAD ===== [OUTPUT] Waktu eksekusi : 5114,03 ms [OUTPUT] Ukuran gambar sebelum : 2171,86 KB (2223988 bytes) [OUTPUT] Ukuran gambar setelah : 435,72 KB (446175 bytes) [OUTPUT] Persentase kompresi : 79,94% [OUTPUT] Kedalaman pohon : 13 [OUTPUT] Banyak simpul : 45637 [OUTPUT] Gambar output disimpan di: C:/test/output4.jpg</pre>
Gambar		

Tabel 5.1.4. Pengujian menggunakan metode MAD (2)

5.1.5 Pengujian 5

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase5.jpg Gambar berhasil dimuat: 4156x2953 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 3 [INPUT] Ambang batas (threshold): 50 [INPUT] Ukuran blok minimum: 16 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output5.jpg Starting compression for: testcase5.jpg</pre>	<pre>===== HASIL KOMPRESI METODE MPD ===== [OUTPUT] Waktu eksekusi : 2303,20 ms [OUTPUT] Ukuran gambar sebelum : 5000,78 KB (5120801 bytes) [OUTPUT] Ukuran gambar setelah : 1119,03 KB (1145885 bytes) [OUTPUT] Persentase kompresi : 77,62% [OUTPUT] Kedalaman pohon : 11 [OUTPUT] Banyak simpul : 226945 [OUTPUT] Gambar output disimpan di: C:\test\output5.jpg</pre>
Gambar		

Tabel 5.1.5. Pengujian menggunakan metode MPD (1)

5.1.6 Pengujian 6

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase6.jpg Gambar berhasil dimuat: 6300x4208 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 3 [INPUT] Ambang batas (threshold): 80 [INPUT] Ukuran blok minimum: 16 [INPUT] Alamat absolut gambar hasil kompresi: Input tidak valid: Path harus absolut. Contoh: /mnt/c/Users/... atau C:/Users/... [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output6.jpg Starting compression for: testcase6.jpg</pre>	<pre>===== HASIL KOMPRESI METODE MPD ===== [OUTPUT] Waktu eksekusi : 4600,53 ms [OUTPUT] Ukuran gambar sebelum : 1642,76 KB (1682188 bytes) [OUTPUT] Ukuran gambar setelah : 686,24 KB (702710 bytes) [OUTPUT] Persentase kompresi : 58,23% [OUTPUT] Kedalaman pohon : 12 [OUTPUT] Banyak simpul : 26561 [OUTPUT] Gambar output disimpan di: C:\test\output6.jpg</pre>
Gambar		

Tabel 5.1.6. Pengujian menggunakan metode MPD (2)

5.1.7 Pengujian 7

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase7.jpg Gambar berhasil dimuat: 8256x5584 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 4 [INPUT] Ambang batas (threshold): 6 [INPUT] Ukuran blok minimum: 32 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output7.jpg Starting compression for: testcase7.jpg</pre>	<pre>===== HASIL KOMPRESI METODE ENTROPY ===== [OUTPUT] Waktu eksekusi : 52237,11 ms [OUTPUT] Ukuran gambar sebelum : 14067,83 KB (14405460 bytes) [OUTPUT] Ukuran gambar setelah : 2810,32 KB (2877771 bytes) [OUTPUT] Persentase kompresi : 80,02% [OUTPUT] Kedalaman pohon : 11 [OUTPUT] Banyak simpul : 103673 [OUTPUT] Gambar output disimpan di: C:\test\output7.jpg</pre>
Gambar		

Tabel 5.1.7. Pengujian menggunakan metode Entropy (1)

5.1.7 Pengujian 8

Tipe	Input	Output
CLI	<pre>[RUN] Running the app... [INPUT] Alamat absolut gambar yang akan dikompresi: C:/test/testcase8.jpg Gambar berhasil dimuat: 3840x2160 Error Measurement Methods: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy [INPUT] Pilih metode (1-4): 4 [INPUT] Ambang batas (threshold): 6 [INPUT] Ukuran blok minimum: 1 [INPUT] Alamat absolut gambar hasil kompresi: C:/test/output8.jpg Starting compression for: testcase8.jpg</pre>	<pre>===== HASIL KOMPRESI METODE ENTROPY ===== [OUTPUT] Waktu eksekusi : 11538,60 ms [OUTPUT] Ukuran gambar sebelum : 15143,93 KB (15507380 bytes) [OUTPUT] Ukuran gambar setelah : 1289,72 KB (1320674 bytes) [OUTPUT] Persentase kompresi : 91,48% [OUTPUT] Kedalaman pohon : 10 [OUTPUT] Banyak simpul : 85037 [OUTPUT] Gambar output disimpan di: C:\test\output8.jpg</pre>
Gambar		

Tabel 5.1.8. Pengujian menggunakan metode Entropy (2)

5.2. Analisis Hasil Percobaan

Hasil percobaan di atas bekerja dengan baik dan setiap gambar berhasil dikompresi dengan nilai ambang atas dan ukuran minimum blok yang variatif, walaupun dihadapi dengan ukuran gambar yang besar. Namun dari segi algoritmanya, metode MAD memiliki sedikit perbedaan dibanding yang lain yang mempengaruhi kompleksitas waktunya.

5.2.1 Analisis Algoritma Metode Variance, MPD, dan Entropy

Inti dari algoritma ada dalam metode buildTree dalam kelas VarianceQuadBuilder, QuadBuilderMPD, dan QuadBuilderEntropy. Metode ini secara rekursif membagi gambar menjadi kuadran hingga memenuhi salah satu kondisi berhenti:

- Blok dikurangi menjadi satu piksel.
- Varians blok berada di bawah ambang batas yang ditentukan, atau jumlah piksel blok kurang dari atau sama dengan ukuran blok minimum tertentu.

Asumsikan skenario terburuk di mana kondisi varians tidak pernah terpenuhi sampai kita mencapai kasus dasar piksel tunggal. Dalam situasi seperti itu, gambar dibagi lagi sebagai berikut:

Untuk bidang gambar berukuran n piksel (di mana n mewakili produk lebar dan tingginya), metode ini membaginya menjadi empat bidang yang lebih kecil, masing-masing berisi sekitar $\frac{n}{4}$ piksel.

Saat menghitung variansi dan MPD, perhitungan dilakukan secara konstan. Walaupun ada *for loop* dalam fungsi menghitung entropy, jumlah iterasi bernilai konstan karena yang diiterasikan merupakan list dengan ukuran konstan (`ListCount = new int[256]`). Oleh karena itu ketiga metode memiliki kompleksitas yang sama ketika menghitung tipe error masing-masing.

Pada setiap simpul internal (selain daun), kode melakukan jumlah pekerjaan yang konstan (menyiapkan indeks, mengumpulkan nilai dari empat turunan, menghitung variansi/MPD/entropy, dll). Pengulangan untuk kompleksitas waktu dalam skenario terburuk ini dapat dinyatakan sebagai:

$$T(n) = 4 \times T\left(\frac{n}{4}\right) + O(1)$$

Dengan Teorema Master, ketika algoritma rekursif membagi masalah menjadi submasalah ukuran $\frac{n}{b}$ dan melakukan $O(n^d)$ bekerja di tiap tingkat dengan $d = 0$ (karena pekerjaan ekstra pada simpul konstan), kompleksitas waktunya adalah:

$$T(n) = O(n^{\log_b a}) = O(n^{\log_4 4}) = O(n)$$

Di dalam blok if ($\text{varians} \leq \text{threshold} \parallel \text{pixelCount} \leq \text{minBlockSize}$), algoritma mengeksekusi perulangan berlapis di atas blok:



```
1 for (int dy = 0; dy < height; dy++) {
2     for (int dx = 0; dx < width; dx++) {
3         if (y + dy < result.getHeight() && x + dx < result.getWidth()) {
4             result.setRGB(x + dx, y + dy, rgb);
5         }
6     }
7 }
```

Biaya untuk satu penggabungan ini beroperasi pada wilayah piksel $width \times height$, yaitu, $O(\text{jumlah piksel})$. Dalam kasus terburuk, setiap piksel gambar ditutupi tepat sekali oleh operasi penggabungan ini (karena blok yang digabungkan membentuk partisi gambar). Oleh karena itu, bahkan jika beberapa node mengeksekusi perulangan ini, biaya keseluruhan di seluruh gambar tetap ada $O(n)$.

Oleh karena itu, kompleksitas waktu terburuk dari implementasi metode variance, MPD, dan entropy adalah $O(n)$, di mana n adalah jumlah total piksel dalam gambar. Kesimpulan ini didasarkan pada perincian rekursif gambar dan fakta bahwa setiap piksel diproses (atau dicat) tepat sekali baik dalam kasus dasar atau sebagai bagian dari wilayah yang digabungkan.

5.2.2 Analisis Algoritma Metode MAD

Sama seperti yang lain, metode ini secara rekursif membagi gambar menjadi kuadran hingga memenuhi salah satu kondisi berhenti. Yang membedakan metode ini dengan sisanya adalah ketika menghitung nilai MAD. Metode ini mengulangi seluruh array piksel untuk wilayah untuk menghitung deviasi absolut rata-rata (MAD), yang $O(n)$ bekerja untuk wilayah itu.



```
1 private double calculateMAD(QuadNodeMAD node, int pixelCount) {
2     if (pixelCount == 0) return 0;
3
4     double meanR = (double) node.rSum / pixelCount;
5     double meanG = (double) node.gSum / pixelCount;
6     double meanB = (double) node.bSum / pixelCount;
7
8     double totalDevR = 0, totalDevG = 0, totalDevB = 0;
9     for (int i = 0; i < pixelCount; i++) {
10         totalDevR += Math.abs(node.rPixels[i] - meanR);
11         totalDevG += Math.abs(node.gPixels[i] - meanG);
12         totalDevB += Math.abs(node.bPixels[i] - meanB);
13     }
14
15     double madR = totalDevR / pixelCount;
16     double madG = totalDevG / pixelCount;
17     double madB = totalDevB / pixelCount;
18
19     return (madR + madG + madB) / 3.0;
20 }
```

Gambar 5.2.1. Calculate MAD mengiterasi seluruh pixel dalam satu blok gambar

Selain itu, metode ini membuat array baru (rPixels, gPixels, bPixels) yang ukurannya sama dengan jumlah piksel di wilayah tersebut (n) pada setiap node. Kemudian menggunakan System.arraycopy untuk menyalin nilai piksel dari setiap turunan ke dalam array ini. Total elemen yang disalin untuk induk sama dengan jumlah ukuran array anak-anak, yaitu $O(n)$ untuk wilayah induk itu.



```
1 System.arraycopy(childNode.rPixels, 0, node.rPixels, i, childNode.rPixels.length);
2 System.arraycopy(childNode.gPixels, 0, node.gPixels, i, childNode.gPixels.length);
3 System.arraycopy(childNode.bPixels, 0, node.bPixels, i, childNode.bPixels.length);
```

Gambar 5.2.1. Metode MAD menggunakan array.copy yang merupakan sebuah loop.

Misal $T(n)$ adalah waktu untuk memproses wilayah dengan n piksel. Dalam kasus terburuk (ketika tidak ada penggabungan awal yang dipicu), setiap wilayah dibagi lagi menjadi 4 subwilayah, masing-masing kira-kira berukuran $n/4$. Hubungan rekursif akan menjadi:

$$T(n) = 4 \times T\left(\frac{n}{4}\right) + f(n)$$

Dimana $f(n)$ mewakili pekerjaan ekstra yang dilakukan setelah panggilan rekursif. Kedua wilayah pekerjaan (array copy dan menghitung MAD) memiliki kompleksitas $O(n)$. Oleh karena itu, dapat dituliskan, $f(n) = O(n)$ untuk relasi di atas:

$$T(n) = 4 \times T\left(\frac{n}{4}\right) + O(n)$$

Berdasarkan Teorema Master, karena $\log_4(4) = 1$ dan $f(n)$ berada pada kelas $O(n^{\log_4(4)}) = O(n)$, maka penyelesaian relasi tersebut:

$$T(n) = O(n \log(n))$$

LAMPIRAN

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4. Mengimplementasi seluruh metode perhitungan error wajib	✓	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan		✓
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error		✓
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar		✓
8. Program dan laporan dibuat (kelompok) sendiri	✓	

Pranala repository

https://github.com/lynaten/Tucil2_13523142/

Daftar Pustaka

- [1] R. Munir, *Algoritma Divide and Conquer – Bagian 1*, Bahan Kuliah IF2211 Strategi Algoritma, Sekolah Teknik Elektro dan Informatika, Institut Teknologi Bandung (ITB), 2025.
- [2] GeeksforGeeks, “Introduction to Divide and Conquer Algorithm,” [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/>. [Accessed: 11-Apr-2025].
- [3] T. Wyork, “Quadtrees for Image Processing,” *Medium*, 2019. [Online]. Available: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>. [Accessed: 11-Apr-2025].