

IF2211 Strategi Algoritma

# **Kompresi Gambar dengan Metode Quadtree: Penerapan Algoritma *Divide and Conquer***

## **Laporan Tugas Kecil**

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



**Oleh :**

**Wardatul Khoiroh 13523001**

**Angelina Efrina Prahastaputri 13523060**

**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**JL. GANESA 10, BANDUNG 40132**

**2025**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I.....</b>	<b>3</b>
1.1 Algoritma Divide and Conquer.....	3
1.2 Kompresi Gambar dengan Metode Quadtree.....	3
<b>BAB II.....</b>	<b>6</b>
2.1. Langkah Penyelesaian Kompresi Gambar dengan Metode Quadtree Menggunakan Algoritma Divide and Conquer.....	6
<b>BAB III.....</b>	<b>7</b>
3.1. Spesifikasi Teknis Program.....	7
3.1.1. Struktur Repository.....	7
3.1.2. Source Code.....	8
<b>BAB IV.....</b>	<b>27</b>
4.1. Analisis Kompleksitas Algoritma Program.....	27
4.2. Hasil dan Analisis Percobaan.....	28
4.2.1. Gambar Sama, Metode Pengukuran Error Berbeda.....	28
4.2.2. Gambar Berbeda, Metode Pengukuran Error Variance.....	36
4.2.3. Gambar Berbeda, Metode Pengukuran Error Mean Absolute Deviation.....	40
4.2.4. Gambar Berbeda, Metode Pengukuran Error Max Pixel Difference.....	44
4.2.5. Gambar Berbeda, Metode Pengukuran Error Entropy.....	48
<b>BAB V.....</b>	<b>53</b>
5.1. Pembuatan GIF Proses Kompresi.....	53
<b>BAB VI.....</b>	<b>56</b>
<b>LAMPIRAN.....</b>	<b>57</b>
<b>REFERENSI.....</b>	<b>58</b>

# BAB I

## DESKRIPSI MASALAH

### 1.1 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah algoritma yang menggunakan pendekatan solusi dengan membagi persoalan yang akan diselesaikan menjadi persoalan yang lebih kecil atau subpersoalan sehingga lebih mudah dipecahkan dan independen. Setelah subpersoalan tersebut diselesaikan, solusinya digabungkan kembali untuk memperoleh solusi keseluruhan.

Algoritma *divide and conquer* dapat dibagi menjadi tiga tahap utama. *Divide*, membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan atau karakteristik yang sama dengan persoalan semula namun berukuran lebih kecil, idealnya setiap upa-persoalan berukuran hampir sama. *Conquer*: menyelesaikan masing-masing upa-persoalan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar. *Combine*: menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula. Algoritma *divide and conquer* lebih natural diungkapkan dalam skema rekursif.

### 1.2 Kompresi Gambar dengan Metode Quadtree



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.

Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan persiapan data; 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
  - b. Threshold variansi
  - c. Minimum *block size*
2. Perhitungan error; setiap blok gambar yang sedang diproses dihitung nilai variansinya menggunakan metode yang dipilih. Metode yang dipilih 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. Metode yang dapat dipilih adalah *variance*, *mean absolute deviation* (MAD), *max pixel difference*, dan *entropy*, (SSIM bonus).
3. Pembagian blok; nilai variansi blok dibandingkan dengan threshold. Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum *block size*, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum *block size*, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok. Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.
4. Normalisasi warna; untuk blok yang tidak lagi dibagi, dilakukan 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. Persentase kompresi juga akan dihitung untuk memberikan gambaran mengenai efisiensi metode kompresi yang digunakan. Persentase kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

## BAB II

### PENYELESAIAN

#### **2.1. Langkah Penyelesaian Kompresi Gambar dengan Metode Quadtree Menggunakan Algoritma *Divide and Conquer***

Berikut adalah langkah-langkah penyelesaian kompresi gambar dengan metode Quadtree menggunakan algoritma *divide and conquer* :

1. Pengguna diminta untuk memasukkan beberapa input: alamat absolut gambar yang ingin dikompresi (contohnya path\image.jpg), pilihan metode perhitungan error yang akan digunakan dalam proses, threshold (dalam range yang sesuai metode perhitungan error yang dipilih), minimum *block size* (tidak lebih besar daripada ukuran gambar asli dan setidaknya bernilai 4), alamat absolut gambar hasil kompresi (contohnya path\compressed.jpg), dan alamat absolut GIF (bonus).
2. Program akan membaca file gambar pada alamat absolut yang dimasukkan pengguna dan menyimpan datanya menjadi suatu objek Matriks yang elemen-elemennya adalah objek Pixel. Objek Pixel pada Matriks mewakili nilai intensitas RGB satu pixel pada gambar.
3. Setelah mendapatkan data Matriks of Pixel dari gambar, program akan melakukan inisialisasi objek *root* Quadtree yang menggunakan parameter Matriks of Pixel tersebut. Objek Quadtree akan terinisialisasi dengan ukuran blok gambar sebesar ukuran matriks, posisi “*start*” di (0, 0), *depth* 0, empat node *child* yang masih null, *isParent* *false* (karena belum ada *child*), dan blok (Matriks) yang akan dibagi diambil dari data blok sebelumnya.
4. Algoritma *divide and conquer* diterapkan menggunakan struktur data Quadtree seperti yang sudah dijelaskan pada langkah sebelumnya dan dilakukan dengan skema rekursif. Untuk setiap proses, program akan mengecek apakah blok dapat dibagi. Jika nilai *error* gambar hasil dari metode perhitungan *error* yang dipilih masih di atas threshold, ukuran blok lebih besar daripada minimum *block size*, dan ukuran blok setelah dibagi tidak kurang dari minimum *block size*, blok akan di-*divide* menjadi 4 dan kemudian blok yang sudah dibagi (subblok) akan di-*conquer* dengan terus melakukan rekursif dengan men-*divide* hingga sampai ke kondisi berhenti yakni salah satu syarat pembagian blok tidak terpenuhi. Jika sudah tidak dapat dibagi lagi, blok akan dinormalisasi warnanya sesuai dengan rata-rata nilai RGB blok.
5. Setelah proses rekursif pembagian blok berhenti sepenuhnya, artinya *divide and conquer* telah menemukan solusi atas setiap upa-persoalan yang ada dan siap untuk

di-*combine*. *Combine* ini juga dilakukan dengan skema rekursif, yakni dengan terus menggabungkan blok-blok dari empat *child* node pada node tersebut hingga seluruhnya berhasil digabungkan menjadi satu blok. Jika sudah, maka hasilnya akan disimpan sebagai gambar pada alamat absolut gambar hasil kompresi (format gambar hasil dapat berbeda dengan format gambar input).

6. Output yang akan didapatkan pengguna adalah gambar hasil kompresi, GIF proses kompresi (bonus), waktu eksekusi kompresi, ukuran gambar sebelum dan sesudah, persentase kompresi, jumlah simpul Quadtree, dan kedalaman Quadtree.

Dengan demikian, langkah-langkah penyelesaian tersebut membantu penulis dalam menyelesaikan kompresi gambar dengan metode Quadtree menggunakan algoritma *divide and conquer*.

## BAB III

### IMPLEMENTASI

#### 3.1. Spesifikasi Teknis Program

##### 3.1.1. Struktur *Repository*

```
└── Tucil2_13523001_13523060
    ├── bin
    │   └── strukturdata
    │       ├── Matriks.class
    │       ├── Pixel.class
    │       ├── ErrorMeasurement.class
    │       ├── Gif.class
    │       ├── ImageCompression.class
    │       ├── InputOutputFile.class
    │       ├── QuadTreeNode.class
    ├── doc
    │   ├── .gitkeep
    │   └── Tucil2_13523001_13523060.pdf
    └── src
        └── strukturdata
            ├── Matriks.java
            ├── Pixel.java
            ├── ErrorMeasurement.java
            ├── Gif.java
            ├── ImageCompression.java
            ├── InputOutputFile.java
            └── QuadTreeNode.java
    └── test
        └── compressed
            ├── bret.png
            ├── hoshinoai.jpg
            ├── itbganesha.jpg
            ├── kochengputih.jpg
            ├── mbakniki.jpg
            ├── pantae.jpg
            └── petrik.jpg
```

```
    └── random_1.png
    └── random_2.png
    └── random_3.png
    └── random_4.png
    └── sunookim.jpg
    └── gifs
        └── bret.gif
        └── hoshinoai.gif
        └── itbganesha.gif
        └── kochengputih.gif
        └── mbakniki.gif
        └── pantae.gif
        └── petrik.gif
        └── random_1.gif
        └── random_2.gif
        └── random_3.gif
        └── random_4.gif
        └── sunookim.gif
    └── original
        └── aihoshino.jpg
        └── backburner.jpg
        └── brat.png
        └── itbgane.jpg
        └── kimsunoo.jpg
        └── kucingputih.jpg
        └── pantai.jpg
        └── patrick.jpg
        └── random.png
    └── README.md
```

### 3.1.2. *Source Code*

1. Matriks.java

```
1 package strukturdata;
2 import java.io.FileWriter;
3 import java.io.IOException;
4 import java.util.Scanner;
5
6 public class Matriks {
7     Scanner scan = new Scanner (System.in);
8     public int baris;
9     public int kolom;
10    public Pixel[][] mat;
11    public int barmin = 0;
12    public int colmin = 0;
13    public int barmax = 100;
14    public int colmax = 100;
15
16    /* Membuat matriks kosong */
17    public Matriks (int baris, int kolom){
18        this.baris = baris; //parameter
19        this.kolom = kolom;
20        this.mat = new Pixel [baris][kolom];
21    }
22
23    /* Salin matriks */
24    public Matriks (Pixel[][] mat){
25        this.baris = mat.length;
26        this.kolom = mat[0].length;
27        this.mat = new Pixel [this.baris][this.kolom];
28
29        for (int i =0; i< this.baris; i++){
30            for (int j=0; j< this.kolom;j++){
31                this.mat[i][j] = mat [i][j];
32            }
33        }
34    }
35
36    /*SELEKTOR*/
37    public int GetFirstIdxBar (Matriks M){
38        return barmin;
39    }
40    public int GetFirstIdxCol (Matriks M){
41        return colmin;
42    }
43    public int GetLastIdxBar (Matriks M){
44        return M.baris-1;
```

```

src > strukturdata > Matriks.java > Language Support for Java(TM) by Red Hat > ConcatVertically(Matriks, Matriks)
 6   public class Matriks {
46     public int GetLastIdxCol (Matriks M){
47       return M.kolom-1;
48     }
49     public Pixel GetElement(int m, int n){
50       return mat[m][n];
51     }
52
53     /* Tampilkan Matriks ke Layar */
54     public void PrintMat (){
55       for (int i=0;i<this.baris;i++){
56         for (int j=0;j<this.kolom;j++){
57           System.out.printf(format:".2f",this.mat[i][j]); //printf karena ada format
58         }
59         System.out.printf(format:"\n");
60       }
61     }
62
63     // Melakukan operasi penjumlahan pada dua buah matriks
64     public static Matriks JumlahMatriks(Matriks M1, Matriks M2){
65       Matriks M = new Matriks(M1.baris, M2.kolom);
66
67       for (int i = 0; i < M1.baris; i++){
68         for (int j = 0; j < M1.kolom; j++){
69           M.mat[i][j] = Pixel.JumlahPixel(M1.mat[i][j], M2.mat[i][j]);
70         }
71       }
72       return M;
73     }
74
75     // Menghitung banyak elemen dalam matriks
76     public static int BanyakElemenMatriks(Matriks M){
77       return M.baris * M.kolom;
78     }
79
80     // Menggabung dua buah matriks secara horizontal
81     public static Matriks ConcatHorizontally(Matriks M, Matriks N){
82       Matriks Out = new Matriks(M.baris, M.kolom+N.kolom); // matriks baru memiliki kolom sebanyak gabungan kolom kedua matriks
83       for (int i = 0; i < Out.baris; i++) {
84         for (int j = 0; j < Out.kolom; j++) {
85           if (j < M.kolom) {
86             Out.mat[i][j] = M.mat[i][j];
87           } else {
88             Out.mat[i][j] = N.mat[i][j - M.kolom];
}

```

```

src > strukturdata > Matriks.java > Language Support for Java(TM) by Red Hat > Matriks > ConcatHorizontally(Matriks, Matriks)
  6   public class Matriks {
  7     public static Matriks ConcatHorizontally(Matriks M, Matriks N){
  8       for (int i = 0; i < Out.baris; i++) {
  9         for (int j = 0; j < Out.kolom; j++) {
 10           if (j < M.kolom) {
 11             Out.mat[i][j] = M.mat[i][j];
 12           } else {
 13             Out.mat[i][j] = N.mat[i][j - M.kolom];
 14           }
 15         }
 16       }
 17       return Out;
 18     }
 19
 20     // Menggabung dua buah matriks secara horizontal
 21     public static Matriks ConcatVertically(Matriks M, Matriks N) {
 22       Matriks Out = new Matriks(M.baris + N.baris, M.kolom); // matriks baru memiliki baris sebanyak gabungan baris kedua matriks
 23       for (int i = 0; i < Out.baris; i++) {
 24         for (int j = 0; j < Out.kolom; j++) {
 25           if (i < M.baris) {
 26             Out.mat[i][j] = M.mat[i][j];
 27           } else {
 28             Out.mat[i][j] = N.mat[i - M.baris][j];
 29           }
 30         }
 31       }
 32       return Out;
 33     }
 34
 35     public static void saveMatrixToText(Matriks M, String filePath) {
 36       try (FileWriter writer = new FileWriter(filePath)) {
 37         for (Pixel[] row : M.mat) {
 38           for (Pixel p : row) {
 39             writer.write("(" + p.getRed() + "," + p.getGreen() + "," + p.getBlue() + ") ");
 40           }
 41           writer.write(str:"\n");
 42         }
 43         System.out.println("Matrix saved to: " + filePath);
 44       } catch (IOException e) {
 45         System.out.println("Error saving matrix to file: " + e.getMessage());
 46       }
 47     }
 48   }
 49 }

```

## 2. Pixel.java

```
src > strukturdata > Pixel.java > Language Support for Java(TM) by Red Hat > {} strukturdata
 1 package strukturdata;
 2
 3 public class Pixel {
 4     int red;
 5     int green;
 6     int blue;
 7
 8     public Pixel(int red, int green, int blue) {
 9         this.red = red;
10         this.green = green;
11         this.blue = blue;
12     }
13
14     public int getRed() {
15         return this.red;
16     }
17
18     public int getGreen() {
19         return this.green;
20     }
21
22     public int getBlue() {
23         return this.blue;
24     }
25
26     public static Pixel JumlahPixel(Pixel p1, Pixel p2) {
27         Pixel p = new Pixel(p1.getRed(), p1.getGreen(), p1.getBlue());
28         p.red += p2.getRed();
29         p.green += p2.getGreen();
30         p.blue += p2.getBlue();
31
32         return p;
33     }
34
35 }
36 }
```

## 3. ErrorMeasurement.java

```

src > ErrorMeasurement.java > Language Support for Java(TM) by Red Hat > ErrorMeasurement > entropy(Matriks)
1   import strukturdata.Matriks;
2
3   public class ErrorMeasurement {
4       public static boolean checkImageError(Matriks matrix, int error_method, double threshold) {
5           double error = 0.0;
6           if (error_method == 1) {
7               error = ErrorMeasurement.variance(matrix);
8           } else if (error_method == 2) {
9               error = ErrorMeasurement.mean_absolute_deviation(matrix);
10          }
11      else if (error_method == 3) {
12          error = ErrorMeasurement.max_pixel_difference(matrix);
13      } else if (error_method == 4) {
14          error = ErrorMeasurement.entropy(matrix);
15      }
16      if (error > threshold) {
17          return true;
18      } else {
19          return false;
20      }
21  }
22  //metode pengukuran error dengan variance
23  public static double variance(Matriks matrix){
24      int panjang = matrix.baris;
25      int lebar = matrix.kolom;
26      int N = panjang*lebar;
27
28      int sum_red = 0;
29      int sum_green = 0;
30      int sum_blue = 0;
31      for (int i=0;i<panjang;i++){
32          for (int j = 0; j < lebar; j++){
33              sum_red += matrix.mat[i][j].getRed();
34              sum_green += matrix.mat[i][j].getGreen();
35              sum_blue += matrix.mat[i][j].getBlue();
36          }
37      }
38      double average_red = (double)sum_red/N;
39      double average_green = (double)sum_green/N;
40      double average_blue = (double)sum_blue/N;
41
42      double a = 0;
43      double b = 0;
44  }

```

```

src > ErrorMeasurement.java > Language Support for Java(TM) by Red Hat > ErrorMeasurement > entropy(Matriks)
 3   public class ErrorMeasurement {
23     public static double variance(Matriks matrix){
45       double c = 0;
46       for (int i=0;i<panjang;i++){
47         for (int j=0;j<lebar;j++){
48           a += Math.pow((matrix.mat[i][j].getRed()-average_red),b:2);
49           b += Math.pow((matrix.mat[i][j].getGreen()-average_green),b:2);
50           c += Math.pow((matrix.mat[i][j].getBlue()-average_blue),b:2);
51         }
52       }
53
54       double variansi_red = a/N;
55       double variansi_green = b/N;
56       double variansi_blue = c/N;
57
58       double variansi_rgb = (variansi_red+variansi_green+variansi_blue)/3;
59       return variansi_rgb;
60     }
61
62     //metode pengukuran error dengan Mean Absolute Deviation (MAD)
63     public static double mean_absolute_deviation(Matriks matrix){
64       int panjang = matrix.baris;
65       int lebar = matrix.kolom;
66       int N = panjang*lebar;
67
68       int sum_red = 0;
69       int sum_green = 0;
70       int sum_blue = 0;
71       for (int i=0;i<panjang;i++){
72         for (int j=0;j<lebar;j++){
73           sum_red += matrix.mat[i][j].getRed();
74           sum_green += matrix.mat[i][j].getGreen();
75           sum_blue += matrix.mat[i][j].getBlue();
76         }
77       }
78
79       double average_red = (double)sum_red/N;
80       double average_green = (double)sum_green/N;
81       double average_blue = (double)sum_blue/N;
82
83       double a = 0;
84       double b = 0;
85       double c = 0;
86       for (int i=0;i<panjang;i++){

```

```

src > 📄 ErrorMeasurement.java > Language Support for Java(TM) by Red Hat > 📄 ErrorMeasurement > ⚙️ entropy(Matriks)
  3   public class ErrorMeasurement {
  63     public static double mean_absolute_deviation(Matriks matrix){
  86       for (int i=0;i<panjang;i++){
  87         for (int j=0;j<lebar;j++){
  88           a += Math.abs(matrix.mat[i][j].getRed()-average_red);
  89           b += Math.abs(matrix.mat[i][j].getGreen()-average_green);
  90           c += Math.abs(matrix.mat[i][j].getBlue()-average_blue);
  91         }
  92       }
  93
  94       double MAD_red = a/N;
  95       double MAD_green = b/N;
  96       double MAD_blue = c/N;
  97
  98       double MAD_rgb = (MAD_red+MAD_green+MAD_blue)/3;
  99       return MAD_rgb;
100     }
101
102    //metode pengukuran error dengan Max Pixel Difference (MPD)
103    public static double max_pixel_difference(Matriks matrix){
104      int panjang = matrix.baris;
105      int lebar = matrix.kolom;
106
107      int min_red = 99999;
108      int max_red = -99999;
109      int min_green = 99999;
110      int max_green = -99999;
111      int min_blue = 99999;
112      int max_blue = -99999;
113
114      for(int i=0;i<panjang;i++){
115        for(int j=0;j<lebar;j++){
116          int red_now = matrix.mat[i][j].getRed();
117          int green_now = matrix.mat[i][j].getGreen();
118          int blue_now = matrix.mat[i][j].getBlue();
119
120          if(red_now>max_red){
121            max_red=red_now;
122          }
123
124          if(red_now<min_red){
125            min_red=red_now;
126          }
127
128          if(green_now>max_green){
129            max_green=green_now;
130          }
131
132          if(green_now<min_green){
133            min_green=green_now;
134          }
135
136        }
137      }
138
139      return (max_red-min_red)/(255*255*3);
140    }
  
```

```

src >  ErrorMeasurement.java > ...
  3   public class ErrorMeasurement {
103     public static double max_pixel_difference(Matriks matrix){
127       if(green_now<min_green){
128         min_green=green_now;
129       }
130       if(blue_now>max_blue){
131         max_blue=blue_now;
132       }
133       if(blue_now<min_blue){
134         min_blue=blue_now;
135       }
136     }
137   }
138
139   int d_red = max_red-min_red;
140   int d_green = max_green-min_green;
141   int d_blue = max_blue-min_blue;
142
143   double d_rgb = (double)(d_red+d_green+d_blue)/3;
144   return d_rgb;
145 }
146
147 //metode pengukuran error dengan Entropy
148 public static double entropy(Matriks matrix){
149   int panjang = matrix.baris;
150   int lebar = matrix.kolom;
151   int N = panjang*lebar;
152
153   double[] freq_red = new double[256];
154   double[] freq_green = new double[256];
155   double[] freq_blue = new double[256];
156
157   for(int i=0;i<panjang;i++){
158     for(int j=0;j<lebar;j++){
159       int red_now = matrix.mat[i][j].getRed();
160       int green_now = matrix.mat[i][j].getGreen();
161       int blue_now = matrix.mat[i][j].getBlue();
162       freq_red[red_now]++;
163       freq_green[green_now]++;
164       freq_blue[blue_now]++;
165     }
166   }
167   for(int i=0;i<256;i++){
168     freq_red[i] = freq_red[i]/N;

```

```

src > ErrorMeasurement.java > ...
  3   public class ErrorMeasurement {
148     public static double entropy(Matriks matrix){
150
157       for(int i=0;i<panjang;i++){
158         for(int j=0;j<lebar;j++){
159           int red_now = matrix.mat[i][j].getRed();
160           int green_now = matrix.mat[i][j].getGreen();
161           int blue_now = matrix.mat[i][j].getBlue();
162           freq_red[red_now]++;
163           freq_green[green_now]++;
164           freq_blue[blue_now]++;
165         }
166       }
167       for(int i=0;i<256;i++){
168         freq_red[i] = freq_red[i]/N;
169         freq_green[i] = freq_green[i]/N;
170         freq_blue[i] = freq_blue[i]/N;
171       }
172
173       double entropy_red = 0;
174       double entropy_green = 0;
175       double entropy_blue = 0;
176       for(int i=0;i<panjang;i++){
177         for(int j=0;j<lebar;j++){
178           int red_now = matrix.mat[i][j].getRed();
179           int green_now = matrix.mat[i][j].getGreen();
180           int blue_now = matrix.mat[i][j].getBlue();
181           if(freq_red[red_now] != 0){
182             entropy_red += freq_red[red_now]*(Math.log(freq_red[red_now])/Math.log(a:2));
183           }
184           if(freq_green[green_now] != 0){
185             entropy_green += freq_green[green_now]*(Math.log(freq_green[green_now])/Math.log(a:2));
186           }
187           if(freq_blue[blue_now] != 0){
188             entropy_blue += freq_blue[blue_now]*(Math.log(freq_blue[blue_now])/Math.log(a:2));
189           }
190         }
191       }
192
193       double entropy_rgb = -(entropy_red+entropy_green+entropy_blue)/3;
194       return entropy_rgb;
195     }
196   }
197 }
```

## 4. ImageCompression.java



```
src > ImageCompression.java > Language Support for Java(TM) by Red Hat > ImageCompression > main(String[])
 5  public class ImageCompression {
 7      public static void main(String[] args) {
42          while (true) {
43              System.out.print("Masukkan pilihan (1-4): ");
44              error_method = scanner.nextInt();
45              if (error_method < 1 || error_method > 4) {
46                  System.out.println("Pilihan tidak valid! Coba lagi.");
47                  continue;
48              } else {
49                  break;
50              }
51          }
52
53          // Meminta input threshold
54          double threshold = 0.0;
55          while (true) {
56              System.out.println("Masukkan ambang batas (threshold) : ");
57              threshold = scanner.nextDouble();
58              if (error_method == 1){
59                  if (threshold < 10 || threshold > 1000) {
60                      System.out.println("Threshold tidak sesuai/invalid! Coba lagi.");
61                      continue;
62                  }
63              }
64              else if (error_method == 2){
65                  if (threshold < 1 || threshold > 10) [
66                      System.out.println("Threshold tidak sesuai/invalid! Coba lagi.");
67                      continue;
68                  ]
69              }
70              else if (error_method == 3){
71                  if (threshold < 5 || threshold > 25) {
72                      System.out.println("Threshold tidak sesuai/invalid! Coba lagi.");
73                      continue;
74                  }
75              }
76              else if (error_method == 4){
77                  if (threshold < 0.5 || threshold > 3.0) {
78                      System.out.println("Threshold tidak sesuai/invalid! Coba lagi.");
79                      continue;
80                  }
81              }
82          }
83      }
84  }
```

```
src > ImageCompression.java > Language Support for Java(TM) by Red Hat > imageCompression > main(String[])
 5  public class ImageCompression {
 7      public static void main(String[] args) {
 85         // Meminta input ukuran blok minimum
 86         double minBlockSize = 0.0;
 87         while (true) {
 88             System.out.println("Masukkan ukuran blok minimum (luas piksel) : ");
 89             minBlockSize = scanner.nextDouble();
 90             scanner.nextLine();
 91             if (minBlockSize < 4 || minBlockSize > matrix.baris || minBlockSize > matrix.kolom) {
 92                 System.out.println("Ukuran minimum blok tidak valid! Coba lagi.");
 93                 continue;
 94             } else {
 95                 break;
 96             }
 97         }
 98
 99         // Meminta alamat output gambar
100        String outputpath;
101        while (true) {
102            System.out.println("Masukkan alamat absolut gambar hasil kompresi : ");
103            outputpath = scanner.nextLine().trim();
104            if (outputpath.isEmpty()) {
105                System.out.println("Alamat gambar tidak boleh kosong! Coba lagi.");
106                continue;
107            } else {
108                break;
109            }
110        }
111
112        // Meminta alamat output gif
113        String output_gif;
114        while (true) {
115            System.out.println("Masukkan alamat absolut output GIF (contoh: path/compression.gif) : ");
116            output_gif = scanner.nextLine().trim();
117            if (output_gif.isEmpty()) {
118                System.out.println("Alamat GIF tidak boleh kosong! Coba lagi.");
119                continue;
120            } else {
121                break;
122            }
123        }
124
125    }
126 }
```

```

src > ImageCompression.java > Language Support for Java(TM) by Red Hat > ImageCompression > main(String[])
 5   public class ImageCompression {
 7     public static void main(String[] args) {
130     System.out.println();
131     System.out.println("=====");
132     System.out.println("Gambar sedang diproses...");
133
134     // Inisialisasi
135     int width = matrix.kolom;
136     int height = matrix.baris;
137     QuadTreeNode root = new QuadTreeNode(matrix, x:0, y:0, width, height, depth:0);
138
139     long start_time = System.currentTimeMillis();
140
141     // Compression
142     QuadTreeNode.divideBlockRecursively(root, matrix, minBlockSize, error_method, threshold); // divide n conquer
143     Matriks compressed = root.createImage(); // combine
144     InputOutputFile.outputFile(compressed, outputpath); // output
145
146     long end_time = System.currentTimeMillis();
147
148
149     System.out.println("Waktu eksekusi: " + (end_time - start_time) + " ms");
150     File original = new File(inputpath);
151     File hasil = new File(outputpath);
152     double persentase = hitungPersentaseKompresi(original, hasil);
153     System.out.printf(format:"Persentase kompresi: %.2f%\n", persentase);
154     System.out.println("Banyak Simpul Quadtree: " + root.countNodes());
155     System.out.println("Kedalaman QuadTree: " + root.getMaxDepth());
156
157     // Membuat GIF dari hasil kompresi per kedalaman
158     int maxDepth = root.getMaxDepth();
159
160     // Membuat GIF
161     Gif.createGif(root, maxDepth, output_gif);
162   }

```

```

src > ImageCompression.java > Language Support for Java(TM) by Red Hat > ImageCompression > main(String[])
 5   public class ImageCompression {
162
163     public static double hitungPersentaseKompresi(File originalFile, File compressedFile) {
164       if (!originalFile.exists() || !compressedFile.exists()) {
165         throw new IllegalArgumentException(s:"Salah satu file tidak ditemukan.");
166       }
167
168       long originalSize = originalFile.length();
169       System.out.println("Ukuran gambar asli: " + originalSize + " bytes");
170       long compressedSize = compressedFile.length();
171       System.out.println("Ukuran gambar hasil kompresi: " + compressedSize + " bytes");
172
173       if (originalSize == 0) {
174         throw new IllegalArgumentException(s:"Ukuran file asli tidak boleh nol.");
175       }
176
177       double ratio = 1.0 - ((double) compressedSize / originalSize);
178       return ratio * 100;
179     }
180   }

```

## 5. InputOutputFile.java

```

src > InputOutputFile.java > Language Support for Java(TM) by Red Hat > InputOutputFile > inputFile()
5   import java.util.Scanner;
6   import javax.imageio.ImageIO;
7   import strukturdata.Matriks;
8   import strukturdata.Pixel;
9
10  public class InputOutputFile {
11
12      // Meminta input alamat absolut gambar
13      public static String inputFile(){
14
15 +     System.out.println("=====");
16     System.out.println();
17
18     Scanner scanner = new Scanner(System.in);
19     String filepath;
20     File file;
21
22     while (true) {
23         System.out.println("Masukkan alamat absolut gambar : ");
24         filepath = scanner.nextLine().trim();
25         if (filepath.isEmpty()) {
26             System.out.println("Alamat gambar tidak boleh kosong! Coba lagi.");
27             continue;
28         }
29         file = new File(filepath);
30         if (!file.exists()) {
31             System.out.println("File tidak ditemukan pada alamat tersebut! Coba lagi.");
32             continue;
33         } else {
34             break;
35         }
36     }
37
38     return filepath;
39
40 }

```

```

src > InputOutputFile.java > Language Support for Java(TM) by Red Hat > InputOutputFile
10  public class InputOutputFile {
11
12      // Membaca file gambar dan menyimpan data pixel pada matriks
13      public static Matriks bacaFileGambar(String filepath) {
14
15         try {
16             File file = new File(filepath);
17             BufferedImage image = ImageIO.read(file);
18             int lebar = image.getWidth(); // lebar dalam pixel
19             int tinggi = image.getHeight(); // tinggi dalam pixel
20             Matriks matriks_pixel = new Matriks(tinggi, lebar);
21
22             // Mengisi matriks dengan nilai intensitas RGB tiap pixel secara horizontal
23             for (int x = 0; x < lebar; x++) {
24                 for (int y = 0; y < tinggi; y++) {
25                     int rgb = image.getRGB(x, y);
26                     Color color = new Color(rgb);
27                     matriks_pixel.mat[y][x] = new Pixel(color.getRed(), color.getGreen(), color.getBlue());
28                 }
29             }
30
31             return matriks_pixel;
32
33         } catch (Exception e) {
34             System.out.println(e.getMessage());
35             return null;
36         }
37     }

```

```
src > InputOutputFile.java > Language Support for Java(TM) by Red Hat > InputOutputFile
 10  public class InputOutputFile {
 11
 12      public static void outputFile(Matriks compressed, String outputpath) {
 13          int height = compressed.baris;
 14          int width = compressed.kolom;
 15          BufferedImage outputImage = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
 16
 17          for (int i = 0; i < height; i++) {
 18              for (int j = 0; j < width; j++) {
 19                  Pixel p = compressed.mat[i][j];
 20                  int rgb = (p.getRed() << 16) | (p.getGreen() << 8) | p.getBlue();
 21                  outputImage.setRGB(j, i, rgb);
 22              }
 23          }
 24
 25          String output_format = "png";
 26          int dotIndex = outputpath.lastIndexOf('.');
 27          if (dotIndex != -1 && dotIndex < outputpath.length() - 1) {
 28              output_format = outputpath.substring(dotIndex + 1).toLowerCase();
 29          }
 30
 31          File outputFile = new File(outputpath);
 32
 33          try {
 34              ImageIO.write(outputImage, output_format, outputFile);
 35              System.out.println("Gambar hasil kompresi berhasil disimpan!");
 36          } catch (IOException e) {
 37              System.err.println("Gagal menyimpan gambar: " + e.getMessage());
 38          }
 39      }
 40
 41  }
```

## 6. QuadTreeNode.java

```

src > QuadTreeNode.java > Language Support for Java(TM) by Red Hat > QuadTreeNode > divideBlock(Matriks, double, int, double)
1  import strukturdata.Matriks;
2  import strukturdata.Pixel;
3  public class QuadTreeNode {
4      Matriks block;
5      int x, y, w, h, depth;
6      private QuadTreeNode q1, q2, q3, q4;
7      private boolean isParent;
8
9      // ctor
10     public QuadTreeNode(Matriks region, int x, int y, int w, int h, int depth) {
11         this.x = x;
12         this.y = y;
13         this.w = w;
14         this.h = h;
15         this.depth = depth;
16         this.q1 = null;
17         this.q2 = null;
18         this.q3 = null;
19         this.q4 = null;
20         this.isParent = false;
21         this.block = getBlockPixels(region, x, y, w, h);
22     }
23
24     public static QuadTreeNode createChildNode (Matriks region, int x, int y, int w, int h, int depth) {
25         return new QuadTreeNode(region, x, y, w, h, depth);
26     }
27
28     public boolean isDivideable (double minblock_size, Matriks region, int error_method, double threshold) {
29         int half_w = this.w / 2;
30         int half_h = this.h / 2;
31         boolean isError = ErrorMeasurement.checkImageError(this.block, error_method, threshold);
32
33         // if (this.w * this.h <= minblock_size) {
34         //     System.out.println("Block size is too small: " + this.w * this.h);
35         // }
36         // if (half_w * half_h <= minblock_size) {
37         //     System.out.println("Half block size is too small: " + half_w * half_h);
38         // }
39         // if (!isError) {
40         //     System.out.println("Error threshold exceeded");
41         // }
42         // System.out.println(" ");
43         return (!this.isParent && this.w * this.h > minblock_size && half_w*half_h >= minblock_size && isError);
44     }
}

```

```

src > QuadTreeNode.java > Language Support for Java(TM) by Red Hat > QuadTreeNode
 3  public class QuadTreeNode {
46    public Matriks getBlockPixels (Matriks region, int x, int y, int w, int h) {
47      Matriks block = new Matriks(h, w);
48      for (int i = 0; i < h; i++) {
49        for (int j = 0; j < w; j++) {
50          block.mat[i][j] = region.mat[y + i][x + j];
51        }
52      }
53      return block;
54    }
55
56    public QuadTreeNode[] divideBlock (Matriks region, double minblock_size, int error_method, double threshold) {
57      if (!isDivideable(minblock_size, region, error_method, threshold)) {
58        return null;
59      }
60      else {
61        this.isParent = true;
62        int half_w1 = this.w / 2;
63        int half_w2 = this.w - half_w1;
64        int half_h1 = this.h / 2;
65        int half_h2 = this.h - half_h1;
66
67        q1 = createChildNode(region, this.x, this.y, half_w1, half_h1, this.depth + 1);
68        q2 = createChildNode(region, this.x + half_w1, this.y, half_w2, half_h1, this.depth + 1);
69        q3 = createChildNode(region, this.x, this.y + half_h1, half_w1, half_h2, this.depth + 1);
70        q4 = createChildNode(region, this.x + half_w1, this.y + half_h1, half_w2, half_h2, this.depth + 1);
71        return new QuadTreeNode[] {q1, q2, q3, q4};
72      }
73    }
74
75    public static void divideBlockRecursively(QuadTreeNode node, Matriks region, double minBlockSize, int error_method, double threshold) {
76      if (node.isDivideable(minBlockSize, region, error_method, threshold) ) {
77        QuadTreeNode[] children = node.divideBlock(region, minBlockSize, error_method, threshold);
78        if (children != null) {
79          for (QuadTreeNode child : children) {
80            divideBlockRecursively(child, region, minBlockSize, error_method, threshold);
81          }
82        }
83      }
84    }
}

```

```

src > QuadTreeNode.java > Language Support for Java(TM) by Red Hat > QuadTreeNode
 3  public class QuadTreeNode {
46    public Matriks createImage() {
47      if (this.isParent && q1 != null && q2 != null && q3 != null && q4 != null) {
48        Matriks image1 = q1.createImage();
49        Matriks image2 = q2.createImage();
50        Matriks image3 = q3.createImage();
51        Matriks image4 = q4.createImage();
52
53        Matriks atas = Matriks.ConcatHorizontally(image1, image2);
54        Matriks bawah = Matriks.ConcatHorizontally(image3, image4);
55        return Matriks.ConcatVertically(atas, bawah);
56      } else {
57        return normalizeBlockColor();
58      }
59    }
60
61    private Matriks normalizeBlockColor() {
62      int totalRed = 0, totalGreen = 0, totalBlue = 0;
63      for (int i = 0; i < h; i++) {
64        for (int j = 0; j < w; j++) {
65          totalRed += this.block.mat[i][j].getRed();
66          totalGreen += this.block.mat[i][j].getGreen();
67          totalBlue += this.block.mat[i][j].getBlue();
68        }
69      }
70      int size = w * h;
71      int meanRed = totalRed / size;
72      int meanGreen = totalGreen / size;
73      int meanBlue = totalBlue / size;
74
75      Matriks N = new Matriks(h, w);
76      for (int i = 0; i < h; i++) {
77        for (int j = 0; j < w; j++) {
78          N.mat[i][j] = new Pixel(meanRed, meanGreen, meanBlue);
79        }
80      }
81      return N;
82    }
83  }
}

```

```

src > QuadTreeNode.java > Language Support for Java(TM) by Red Hat > QuadTreeNode
 3   public class QuadTreeNode {
124     public int countNodes() {
125       if (!this.isParent) {
126         return 1;
127       } else {
128         return 1 + q1.countNodes() + q2.countNodes() + q3.countNodes() + q4.countNodes();
129       }
130     }
131
132     public int getMaxDepth() {
133       if (!this.isParent) {
134         return this.depth;
135       } else {
136         int d1 = q1.getMaxDepth();
137         int d2 = q2.getMaxDepth();
138         int d3 = q3.getMaxDepth();
139         int d4 = q4.getMaxDepth();
140         return Math.max(Math.max(d1, d2), Math.max(d3, d4));
141       }
142     }
143
144     public Matriks createImageAtDepth(int targetDepth) {
145       if (this.depth == targetDepth) {
146         // Jika kedalaman saat ini sama dengan target, gunakan warna rata-rata blok
147         return normalizeBlockColor();
148       } else if (this.isParent) {
149         // Rekursif untuk setiap child, jika child null gunakan warna dari node saat ini
150         Matriks image1 = (q1 != null) ? q1.createImageAtDepth(targetDepth) : createFallbackBlock(this.h / 2, this.w / 2);
151         Matriks image2 = (q2 != null) ? q2.createImageAtDepth(targetDepth) : createFallbackBlock(this.h / 2, this.w - this.w / 2);
152         Matriks image3 = (q3 != null) ? q3.createImageAtDepth(targetDepth) : createFallbackBlock(this.h - this.h / 2, this.w / 2);
153         Matriks image4 = (q4 != null) ? q4.createImageAtDepth(targetDepth) : createFallbackBlock(this.h - this.h / 2, this.w - this.w / 2);
154
155         // Gabungkan matriks dari keempat kuadran
156         Matriks atas = Matriks.ConcatHorizontally(image1, image2);
157         Matriks bawah = Matriks.ConcatHorizontally(image3, image4);
158         return Matriks.ConcatVertically(atas, bawah);
159       } else {
160         // Jika tidak ada child, gunakan warna dari lapisan terakhir yang tersedia
161         return normalizeBlockColor();
162       }
163     }
164   }

```

```

src > QuadTreeNode.java > Language Support for Java(TM) by Red Hat > QuadTreeNode
 3   public class QuadTreeNode {
104
165     private Matriks createFallbackBlock(int height, int width) {
166       // Membuat blok fallback dengan warna rata-rata dari node saat ini
167       Matriks fallback = new Matriks(height, width);
168       Pixel averageColor = new Pixel(
169         this.block.mat[0][0].getRed(),
170         this.block.mat[0][0].getGreen(),
171         this.block.mat[0][0].getBlue()
172       );
173       for (int i = 0; i < height; i++) {
174         for (int j = 0; j < width; j++) {
175           fallback.mat[i][j] = averageColor;
176         }
177       }
178     }
179   }
180 }

```

## BAB IV

### ANALISIS DAN PENGUJIAN

#### **4.1. Analisis Kompleksitas Algoritma Program**

##### **1. Kompleksitas Divide (Pembagian Blok)**

Setiap blok dibagi menjadi empat sub-blok jika memenuhi syarat pembagian. Dalam kasus terburuk, seluruh gambar dibagi hingga setiap piksel menjadi blok tersendiri. Dengan ukuran gambar NxN, jumlah simpul maksimum dalam quadtree adalah  $T(n) = 1 + 4 + 4^2 + \dots + 4^{\log_4(N)} = \frac{4^{\log_4(N)+1} - 1}{3}$ . Karena N adalah jumlah piksel maka kompleksitas waktu untuk pembagian blok adalah O(N<sup>2</sup>) dalam kasus terburuk.

##### **2. Kompleksitas Perhitungan Error**

Error setiap blok dihitung menggunakan salah satu dari empat metode (variance, mean absolute deviation, max pixel difference, atau entropy). Kompleksitas penghitungan error untuk satu blok adalah O(k), di mana k adalah jumlah piksel dalam blok. Dalam kasus terburuk, error dihitung untuk setiap piksel, sehingga kompleksitas total adalah O(N<sup>2</sup>).

##### **3. Kompleksitas Combine (Penggabungan Blok)**

Gambar direkonstruksi dengan menggabungkan blok-blok dari Quadtree. Proses penggabungan dilakukan secara rekursif, dengan kompleksitas O(N<sup>2</sup>) karena setiap piksel diproses sekali.

##### **4. Kompleksitas Total**

Kompleksitas total algoritma adalah: O(N<sup>2</sup>) + O(N<sup>2</sup>) + O(N<sup>2</sup>) = O(N<sup>2</sup>). Algoritma ini efisien untuk gambar dengan ukuran kecil hingga sedang, tetapi dapat menjadi lambat untuk gambar beresolusi tinggi.

## 4.2. Hasil dan Analisis Percobaan

### 4.2.1. Gambar Sama, Metode Pengukuran Error Berbeda



1. Test Case 1
  - Input

- Output

```
=====  
Gambar sedang diproses...  
Gambar hasil kompresi berhasil disimpan!  
Waktu eksekusi: 656 ms  
Ukuran gambar asli: 67167 bytes  
Ukuran gambar hasil kompresi: 82855 bytes  
Persentase kompresi: -23.36%  
Banyak Simpul Quadtree: 1805  
Kedalaman QuadTree: 6  
GIF berhasil dibuat!
```



[GIF1](#)

- **Analisis**

Gambar terkompress dengan persentase -23.36%, hal ini bisa saja terjadi karena threshold terlalu kecil atau minimum block size terlalu kecil sehingga jumlah simpul meningkat drastis. Selain itu, dapat disebabkan karena gambar asli sudah sangat sederhana atau sedikit variasi warna, maka membagi-bagi blok justru menambah informasi tanpa pengurangan ukuran yang signifikan. Ada bagian kabel yang hilang karena threshold masih bisa lebih kecil sehingga untuk blok tersebut sudah mencapai thresholdnya dan tidak membagi lagi.

2. Test Case 2

- Input

```

=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\random.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 2
Masukkan ambang batas (threshold) :
8
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\random_2.png
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\random_2.gif
=====
```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 577 ms
Ukuran gambar asli: 67167 bytes
Ukuran gambar hasil kompresi: 62839 bytes
Persentase kompresi: 6.44%
Banyak Simpul Quadtree: 1161
Kedalaman QuadTree: 6
GIF berhasil dibuat!
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █
```



GIF2

- **Analisis**

Gambar terkompress dengan persentase 6.44%, ada bagian kabel yang hilang karena threshold masih bisa lebih kecil sehingga untuk blok tersebut sudah mencapai thresholdnya dan tidak membagi lagi. Jika ingin mendapatkan hasil kompresi yang mendekati gambar asli maka perlu threshold yang lebih kecil dan minimum block size yang lebih kecil.

3. Test Case 3

- Input

```

=====
[Random Binary Data]
=====

Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\random.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 3
Masukkan ambang batas (threshold) :
20
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\random_3.png
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\random_3.gif

```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 704 ms
Ukuran gambar asli: 67167 bytes
Ukuran gambar hasil kompresi: 102046 bytes
Persentase kompresi: -51.93%
Banyak Simpul Quadtree: 2501
Kedalaman QuadTree: 6
GIF berhasil dibuat!
◆ PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █

```



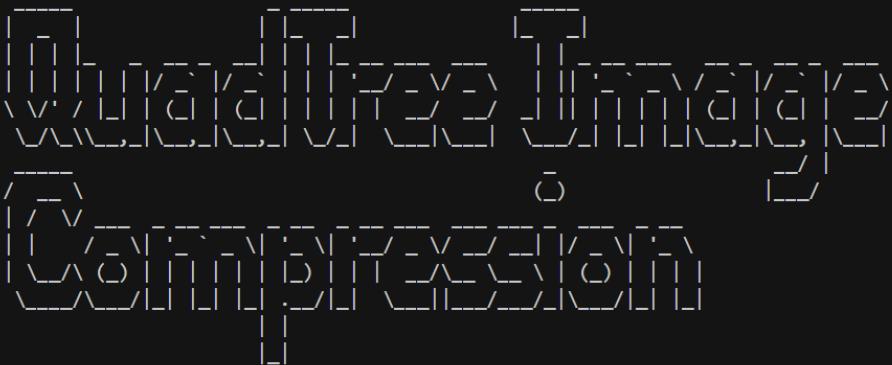
GIF3

- **Analisis**

Gambar terkompress dengan persentase -51.93%. Hasil kompresi sudah bagus hampir mendekati gambar asli dengan ada beragam ukuran block sesuai pendefinisian. Untuk mengecek apakah sebuah method error berhasil, perlu dilihat jumlah nodenya, apabila jumlah node berbeda-beda dengan depth sama maka pembagian berhasil yang artinya pembagian berhenti berdasarkan threshold.

4. Test Case 4

- Input



```

=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\random.png
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 4
Masukkan ambang batas (threshold) :
2
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\random_4.png
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\random_4.gif

```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 1631 ms
Ukuran gambar asli: 67167 bytes
Ukuran gambar hasil kompresi: 158200 bytes
Persentase kompresi: -135.53%
Banyak Simpul Quadtree: 5461
Kedalaman QuadTree: 6
GIF berhasil dibuat!
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █

```



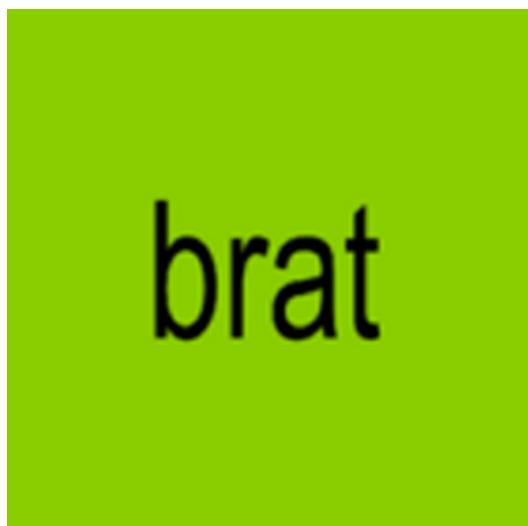
GIF4

- **Analisis**

Gambar terkompress dengan persentase -135.53%. Semakin kecil threshold, ukuran hasil kompresi semakin besar sehingga persentase kompresi menurun dan menyebabkan negatif (tergantung gambar juga). Untuk mengecek apakah sebuah method error berhasil, perlu dilihat jumlah nodenya, apabila jumlah node berbeda-beda dengan depth sama maka pembagian berhasil yang artinya pembagian berhenti berdasarkan threshold.

#### 4.2.2. Gambar Berbeda, Metode Pengukuran Error *Variance*

1. Test Case 5

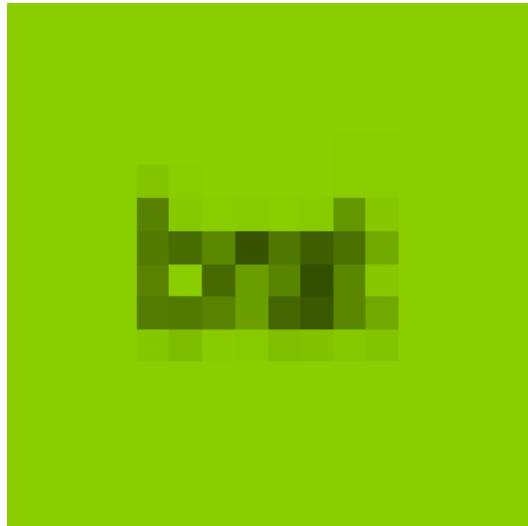


- **Input**

```
=====  
Masukkan alamat absolut gambar :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\brat.png  
Pilih metode perhitungan error:  
1. Variance  
2. Mean Absolute Difference  
3. Max Pixel Difference  
4. Entropy  
Masukkan pilihan (1-4): 1  
Masukkan ambang batas (threshold) :  
20  
Masukkan ukuran blok minimum (luas piksel) :  
100  
Masukkan alamat absolut gambar hasil kompresi :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\bret.png  
Masukkan alamat absolut output GIF (contoh: path/compression.gif):  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\bret.gif
```

- Output

```
=====  
Gambar sedang diproses...  
Gambar hasil kompresi berhasil disimpan!  
Waktu eksekusi: 144 ms  
Ukuran gambar asli: 14801 bytes  
Ukuran gambar hasil kompresi: 4069 bytes  
Persentase kompresi: 72.51%  
Banyak Simpul Quadtree: 89  
Kedalaman QuadTree: 4  
GIF berhasil dibuat!  
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █
```



GIF5

- **Analisis**

Gambar terkompress dengan persentase 72.51%. Dengan threshold dan min block size seperti ini, program berhasil mencapai kompresi optimal dengan kecepatan tinggi dan struktur pohon sederhana. Untuk mengecek apakah sebuah method error berhasil, perlu dilihat jumlah nodenya, apabila jumlah node berbeda-beda dengan depth sama maka pembagian berhasil yang artinya pembagian berhenti berdasarkan threshold.

2. Test Case 6



- **Input**



```

=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\pantai.jpg
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 1
Masukkan ambang batas (threshold) :
20
Masukkan ukuran blok minimum (luas piksel) :
200
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\pantae.jpg
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\pantae.gif

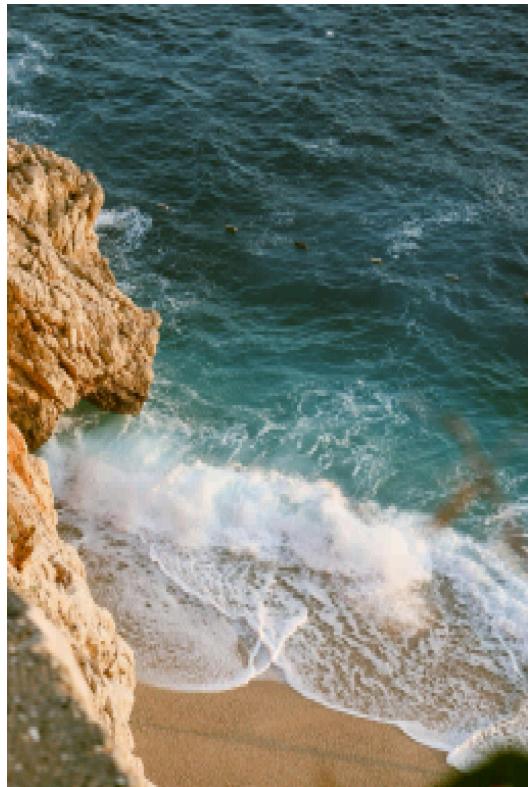
```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 11004 ms
Ukuran gambar asli: 3718573 bytes
Ukuran gambar hasil kompresi: 1485870 bytes
Persentase kompresi: 60.04%
Banyak Simpul Quadtree: 85421
Kedalaman QuadTree: 8
GIF berhasil dibuat!
❖ PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █

```



[GIF6](#)

- **Analisis**

Gambar terkompress dengan persentase 60.04%. Dengan threshold dan min block size seperti ini, program berhasil mencapai kompresi optimal. Dimana tidak terlihat seperti hasil kompresi hal ini menandakan threshold yang semakin kecil membuat kualitas visual hasil kompresi lebih baik. Terlihat ukuran blok seragam karena warnanya yang sangat beragam sehingga error masih lebih besar dari threshold dan proses kompresi berhenti saat mencapai minimum block size.

#### 4.2.3. Gambar Berbeda, Metode Pengukuran Error *Mean Absolute Deviation*

1. Test Case 7

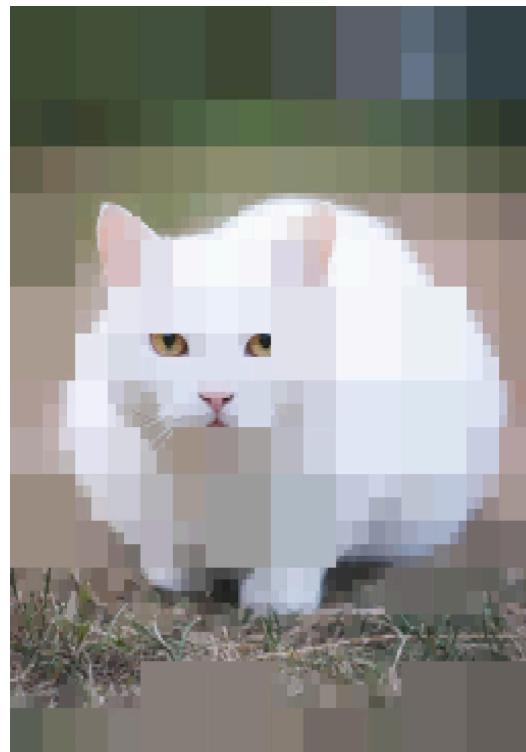


- Input

```
=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\kucingputih.jpg
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 2
Masukkan ambang batas (threshold) :
15
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\kochengputih.jpg
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\kochengputih.gif
```

- Output

```
=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 3298 ms
Ukuran gambar asli: 926737 bytes
Ukuran gambar hasil kompresi: 377767 bytes
Persentase kompresi: 59.24%
Banyak Simpul Quadtree: 7229
Kedalaman QuadTree: 8
GIF berhasil dibuat!
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █
```



[GIF7](#)

- Analisis

Gambar terkompress dengan persentase 59.24%. Hasil kompresi sudah bagus hampir mendekati gambar asli dengan ada beragam ukuran block sesuai pendefinisian.

## 2. Test Case 8



- Input

```
[REDACTED]  
=====  
Masukkan alamat absolut gambar :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\kimsunoo.jpg  
Pilih metode perhitungan error:  
1. Variance  
2. Mean Absolute Difference  
3. Max Pixel Difference  
4. Entropy  
Masukkan pilihan (1-4): 2  
Masukkan ambang batas (threshold) :  
8  
Masukkan ukuran blok minimum (luas piksel) :  
100  
Masukkan alamat absolut gambar hasil kompresi :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\sunookim.jpg  
Masukkan alamat absolut output GIF (contoh: path/compression.gif):  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\sunookim.gif
```

- Output

```
=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 922 ms
Ukuran gambar asli: 153795 bytes
Ukuran gambar hasil kompresi: 82946 bytes
Persentase kompresi: 46.07%
Banyak Simpul Quadtree: 4133
Kedalaman QuadTree: 7
GIF berhasil dibuat!
```



GIF8

- **Analisis**

Gambar terkompress dengan persentase 46.07%. Hasil kompresi sudah bagus hampir mendekati gambar asli dengan ada beragam ukuran block sesuai pendefinisian. Hal ini menandakan bahwa program tidak hanya berpacu pada gambar square, namun pada gambar seperti di atas dapat ditangani dengan baik. Threshold yang kecil menyebabkan kualitas visual hasil kompresi baik.

#### 4.2.4. Gambar Berbeda, Metode Pengukuran Error *Max Pixel Difference*

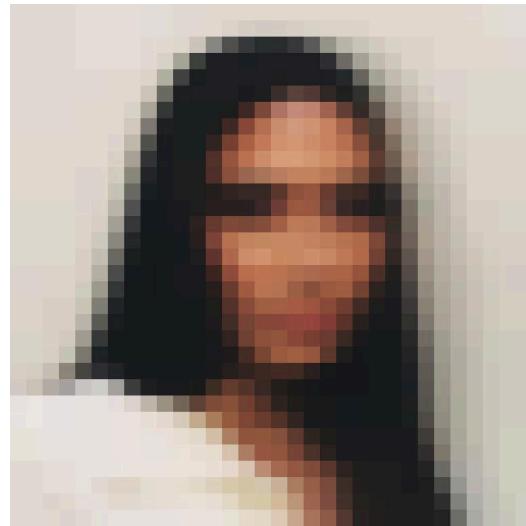


1. Test Case 9
  - Input

```
[REDACTED]  
=====  
Masukkan alamat absolut gambar :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\backburner.jpg  
Pilih metode perhitungan error:  
1. Variance  
2. Mean Absolute Difference  
3. Max Pixel Difference  
4. Entropy  
Masukkan pilihan (1-4): 3  
Masukkan ambang batas (threshold) :  
20  
Masukkan ukuran blok minimum (luas piksel) :  
100  
Masukkan alamat absolut gambar hasil kompresi :  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\mbakniki.jpg  
Masukkan alamat absolut output GIF (contoh: path/compression.gif):  
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\mbakniki.gif
```

- Output

```
=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 312 ms
Ukuran gambar asli: 64392 bytes
Ukuran gambar hasil kompresi: 17576 bytes
Persentase kompresi: 72.70%
Banyak Simpul Quadtree: 973
Kedalaman QuadTree: 5
GIF berhasil dibuat!
```



GIF9

- **Analisis**

Gambar terkompress dengan persentase 72.70%. Hasil kompresi sudah bagus hampir mendekati gambar asli dengan ada beragam ukuran block sesuai pendefinisian. Threshold yang kecil menyebabkan kualitas visual hasil kompresi baik.

2. Test Case 10



- **Input**

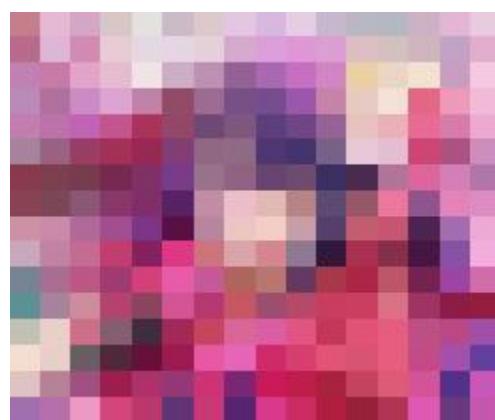
```

=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\aihoshino.jpg
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 3
Masukkan ambang batas (threshold) :
20
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\hoshinoai.jpg
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\hoshinoai.gif
=====
```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 127 ms
Ukuran gambar asli: 12808 bytes
Ukuran gambar hasil kompresi: 7676 bytes
Persentase kompresi: 40.07%
Banyak Simpul Quadtree: 341
Kedalaman QuadTree: 4
GIF berhasil dibuat!
❖ PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █
```



GIF10

- **Analisis**

Gambar terkompress dengan persentase 40.07%. Dengan threshold dan min block size seperti ini, program berhasil mencapai kompresi optimal, namun gambar tidak mendekati gambar asli. Terlihat ukuran blok seragam karena warnanya yang sangat beragam sehingga error masih lebih besar dari threshold dan proses kompresi berhenti saat mencapai minimum block size.

#### 4.2.5. Gambar Berbeda, Metode Pengukuran Error *Entropy*

1. Test Case 11



- Input

```
=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\patrick.jpg
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 4
Masukkan ambang batas (threshold) :
2
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\petrik.jpg
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\petrik.gif
```

- Output

```
=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 726 ms
Ukuran gambar asli: 49540 bytes
Ukuran gambar hasil kompresi: 43724 bytes
Persentase kompresi: 11.74%
Banyak Simpul Quadtree: 5381
Kedalaman QuadTree: 6
GIF berhasil dibuat!
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060>
```



GIF11

- **Analisis**

Gambar terkompress dengan persentase 11.74%. Dengan threshold dan min block size seperti ini, program berhasil mencapai kompresi optimal. Dimana tidak terlihat seperti hasil kompresi hal ini menandakan threshold yang semakin kecil membuat kualitas visual hasil kompresi lebih baik. Terlihat ukuran blok seragam karena warnanya yang sangat beragam sehingga error masih lebih besar dari threshold dan proses kompresi berhenti saat mencapai minimum block size.

2. Test Case 12



- **Input**

```

=====
Masukkan alamat absolut gambar :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\original\itbgane.jpg
Pilih metode perhitungan error:
1. Variance
2. Mean Absolute Difference
3. Max Pixel Difference
4. Entropy
Masukkan pilihan (1-4): 4
Masukkan ambang batas (threshold) :
2
Masukkan ukuran blok minimum (luas piksel) :
100
Masukkan alamat absolut gambar hasil kompresi :
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\compressed\itbganesha.jpg
Masukkan alamat absolut output GIF (contoh: path/compression.gif):
C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060\test\gifs\itbganesha.gif
=====
```

- Output

```

=====
Gambar sedang diproses...
Gambar hasil kompresi berhasil disimpan!
Waktu eksekusi: 11437 ms
Ukuran gambar asli: 3539130 bytes
Ukuran gambar hasil kompresi: 753543 bytes
Persentase kompresi: 78.71%
Banyak Simpul Quadtree: 87373
Kedalaman QuadTree: 8
GIF berhasil dibuat!
PS C:\Users\Lenovo\Documents\Stima\Tucil2_13523001_13523060> █
```



GIF12

- Analisis

Gambar terkompress dengan persentase 78.71%. Dengan threshold dan min block size seperti ini, program berhasil mencapai kompresi optimal. Dimana tidak terlihat seperti hasil kompresi hal ini menandakan threshold yang semakin kecil membuat kualitas visual hasil kompresi lebih baik. Terlihat ukuran blok seragam karena warnanya yang sangat beragam sehingga error masih lebih besar dari threshold dan proses kompresi berhenti saat mencapai minimum block size.

## BAB V

# IMPLEMENTASI BONUS

### 5.1. Pembuatan GIF Proses Kompresi

Pada bonus ini, proses pembagian/divide blok dalam pembentukan quadtree divisualisasikan dengan format GIF. Visualisasi dilakukan per depth dimana setiap pembagian quadtree menjadi anak atau depth bertambah maka frame dibentuk sehingga nantinya bersatu dan terbentuk gif. Proses animasi dimulai dengan menampilkan proses kompresi gambar pada depth = 0 hingga maxDepth. Pada tiap iterasi membentuk frame dengan mengecek apakah kedalaman tersebut merupakan target kedalaman yang diminta sehingga menghasilkan gambar berdasarkan rata-rata warna bloknya. Selain itu, mengecek juga apabila kedalaman tersebut masih memiliki anak (parent) maka dilakukan iterasi hingga tidak dapat dibagi lagi (kurang dari threshold). Apabila bagian matriks tersebut tidak dapat dibagi lagi namun masih terdapat bagian matriks lain yang dapat dibagi, maka bagian matriks yang tidak dapat dibagi lagi dibuat frame dengan warna rata-rata dari node saat ini sehingga gif tetap berjalan (node tidak menjadi hitam/blank). Setelah pengecekan tersebut, setiap bagian matriks digabungkan kembali secara horizontal dan vertikal sehingga terbentuk matriks utuh dengan ukuran elemennya beragam pada tiap kedalaman. Matriks perlu diubah menjadi bufferedimage yang bisa diproses oleh Java ImageIO. Kemudian, perlu membuat metadata frame GIF dengan delay antar frame, disposal method, dan looping behavior sehingga memungkinkan GIF beranimasi otomatis dan kecepatan animasi dapat dikendalikan. Setelahnya, tambahkan setiap gambar yang sudah dikonversi dan diberi metadata ke urutan frame dalam GIF. Algoritma mengenai GIF sendiri berada pada file Gif.java dengan bantuan fungsi `createImageAtDepth` dan `createFallbackBlock` pada file QuadTreeNode.java. Berikut implementasi algoritma GIF dalam Bahasa Java:

1. `createImageAtDepth(int targetDepth)` : untuk membentuk representasi gambar hasil kompresi pada kedalaman tertentu (depth) dari pohon quadtree.
2. `createFallbackBlock(int height, int width)` : untuk menghasilkan blok gambar berukuran tertentu berisi warna rata-rata node sata ini (digunakan saat anak null).
3. `createGif(QuadTreeNode root, int maxDepth, String outputPath)` : untuk membuat GIF animasi dari hasil kompresi quadtree dari berbagai kedalaman (fungsi utama).
4. `convertMatriksToBufferedImage(Matriks matriks)` : untuk konversi representasi internal (Matriks dari Pixel[][])) menjadi BufferedImage Java.
5. `configureMetadata(IIMetadata metadata, int delayTime)` : untuk mengatur metadata animasi frame GIF.

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.imageio.stream.ImageOutputStream;
import javax.imageio.ImageWriter;
import javax.imageio.metadata.IIOMetadata;
import javax.imageio.metadata.IIOMetadataNode;
import javax.imageio.IIOImage;
import strukturdata.Matriks;
import strukturdata.Pixel;

You, 2 hours ago | 1 author (You)
public class Gif {
    public static void createGif(QuadTreeNode root, int maxDepth, String outputPath) {
        try (ImageOutputStream output = ImageIO.createImageOutputStream(new File(outputPath))) {
            ImageWriter gifWriter = ImageIO.getImageWritersByFormatName("gif").next();
            gifWriter.setOutput(output);

            // Konfigurasi metadata untuk GIF animasi
            gifWriter.prepareWriteSequence(streamMetadata:null);

            for (int depth = 0; depth <= maxDepth; depth++) {
                Matriks imageAtDepth = root.createImageAtDepth(depth);
                BufferedImage bufferedImage = convertMatriksToBufferedImage(imageAtDepth);

                // Buat metadata untuk frame
                IIOMetadata metadata = gifWriter.getDefaultImageMetadata(new javax.imageio.ImageTypeSpecifier(bufferedImage), param:null);
                configureMetadata(metadata, delayTime:500); // Delay 500ms per frame

                // Tulis frame ke GIF
                gifWriter.writeToSequence(new IIOImage(bufferedImage, thumbnails:null, metadata), param:null);
            }

            gifWriter.endWriteSequence();
            System.out.println("GIF berhasil dibuat di: " + outputPath);
        }
    }

    public class QuadTreeNode {
        public Matriks createImageAtDepth(int targetDepth) {
            if (this.depth == targetDepth) {
                // Jika kedalaman saat ini sama dengan target, gunakan warna rata-rata blok
                return normalizeBlockColor();
            } else if (this.isParent) {
                // Rekursif untuk setiap child, jika child null gunakan warna dari node saat ini
                Matriks image1 = (q1 != null) ? q1.createImageAtDepth(targetDepth) : createFallbackBlock(this.h / 2, this.w / 2);
                Matriks image2 = (q2 != null) ? q2.createImageAtDepth(targetDepth) : createFallbackBlock(this.h / 2, this.w - this.w / 2);
                Matriks image3 = (q3 != null) ? q3.createImageAtDepth(targetDepth) : createFallbackBlock(this.h - this.h / 2, this.w / 2);
                Matriks image4 = (q4 != null) ? q4.createImageAtDepth(targetDepth) : createFallbackBlock(this.h - this.h / 2, this.w - this.w / 2);

                // Gabungkan matriks dari keempat kuadran
                Matriks atas = Matriks.ConcatHorizontally(image1, image2);
                Matriks bawah = Matriks.ConcatHorizontally(image3, image4);
                return Matriks.ConcatVertically(atas, bawah);
            } else {
                // Jika tidak ada child, gunakan warna dari lapisan terakhir yang tersedia
                return normalizeBlockColor();
            }
        }

        private Matriks createFallbackBlock(int height, int width) {
            // Membuat blok fallback dengan warna rata-rata dari node saat ini
            Matriks fallback = new Matriks(height, width);
            Pixel averageColor = new Pixel([
                this.block.mat[0][0].getRed(),
                this.block.mat[0][0].getGreen(),
                this.block.mat[0][0].getBlue()
            ]);
            for (int i = 0; i < height; i++) {
                for (int j = 0; j < width; j++) {
                    fallback.mat[i][j] = averageColor;
                }
            }
            return fallback;
        }
    }
}

```

```

        } catch (IOException e) {
            System.err.println("Gagal membuat GIF: " + e.getMessage());
        }
    }

    private static BufferedImage convertMatriksToBufferedImage(Matriks matriks) {
        int width = matriks.kolom;
        int height = matriks.baris;
        BufferedImage image = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);

        for (int i = 0; i < height; i++) {
            for (int j = 0; j < width; j++) {
                Pixel p = matriks.mat[i][j];
                if (p == null) {
                    // Jika elemen null, gunakan warna default (hitam)
                    image.setRGB(j, i, rgb:0);
                } else {
                    int rgb = (p.getRed() << 16) | (p.getGreen() << 8) | p.getBlue();
                    image.setRGB(j, i, rgb);
                }
            }
        }

        return image;
    }

    private static void configureMetadata(IIOMetadata metadata, int delayTime) {
        String metaFormatName = metadata.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode) metadata.getAsTree(metaFormatName);

        IIOMetadataNode graphicsControlExtensionNode = getNode(root, nodeName:"GraphicControlExtension");
        graphicsControlExtensionNode.setAttribute(name:"disposalMethod", value:"none");
        graphicsControlExtensionNode.setAttribute(name:"userInputFlag", value:"FALSE");
        graphicsControlExtensionNode.setAttribute(name:"transparentColorFlag", value:"FALSE");

graphicsControlExtensionNode.setAttribute(name:"delayTime", Integer.toString(delayTime / 10)); // Delay dalam 1/100 detik
graphicsControlExtensionNode.setAttribute(name:"transparentColorIndex", value:"0");

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

        try {
            metadata.setFromTree(metaFormatName, root);
        } catch (Exception e) {
            throw new RuntimeException("Gagal mengatur metadata GIF: " + e.getMessage());
        }
    }

    private static IIOMetadataNode getNode(IIOMetadataNode rootNode, String nodeName) {
        for (int i = 0; i < rootNode.getLength(); i++) {
            if (rootNode.item(i).getNodeName().equalsIgnoreCase(nodeName)) {
                return (IIOMetadataNode) rootNode.item(i);
            }
        }
        IIOMetadataNode node = new IIOMetadataNode(nodeName);
        rootNode.appendChild(node);
        return node;
    }
}

```

## BAB VI

### KESIMPULAN

Berdasarkan analisis dan implementasi algoritma Divide and Conquer dalam kompresi gambar menggunakan metode Quadtree, dapat disimpulkan bahwa algoritma ini efektif dalam merepresentasikan gambar secara hierarkis dengan mempertimbangkan kompleksitas lokal tiap blok gambar. Dengan membagi gambar secara rekursif menjadi kuadran dan menghentikan pembagian berdasarkan batasan error, threshold, dan ukuran minimum blok, algoritma ini mampu menghasilkan kompresi yang adaptif terhadap konten gambar. Eksperimen menunjukkan bahwa algoritma bekerja sangat efisien untuk gambar dengan sedikit variasi warna atau area homogen, menghasilkan pohon yang lebih dangkal dan waktu eksekusi yang singkat. Sebaliknya, untuk gambar dengan variasi warna yang tinggi dan detail halus, struktur pohon menjadi lebih dalam dan kompleks, sehingga membutuhkan waktu eksekusi yang lebih lama dan ruang penyimpanan pohon yang lebih besar. Empat metode pengukuran error (seperti variance, MAD, MPD, dan entropy) memberikan fleksibilitas dalam menyesuaikan kualitas hasil kompresi terhadap karakteristik gambar. Hal ini memungkinkan pengguna memilih pendekatan yang paling sesuai dengan tujuan akhir, apakah untuk efisiensi penyimpanan, kualitas visual, atau analisis struktur. Secara keseluruhan, algoritma Quadtree berbasis Divide and Conquer menawarkan kombinasi efisiensi, skalabilitas, dan kontrol kualitas yang baik dalam proses kompresi gambar. Pemahaman yang mendalam terhadap parameter algoritma dan karakteristik input sangat penting dalam mengoptimalkan performa dan hasil akhir kompresi.

## LAMPIRAN

**Repository Github:**

[https://github.com/angelinaefrina/Tucil2\\_13523001\\_13523060.git](https://github.com/angelinaefrina/Tucil2_13523001_13523060.git)

**Tabel Pemeriksaan:**

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	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	

## REFERENSI

<https://www.scitepress.org/papers/2013/42105/42105.pdf>

<https://wwwodelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics/>

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

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-\(69\).pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2022-2023/Makalah/Makalah-Stima-2023-(69).pdf)