

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:

Muhammad Aufa Farabi 13523023

Joel Hotlan Haris Siahaan 13523025

Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

DAFTAR ISI.....	1
BAGIAN I.....	2
ALGORITMA DIVIDE AND CONQUER.....	2
BAGIAN II.....	3
SOURCE PROGRAM.....	3
BAGIAN III.....	20
SCREENSHOT HASIL TEST.....	20
3.1 Input file: flower.jpg.....	20
3.2 Input file: trump.jpg.....	21
3.3 Input file: messi.jpg.....	23
3.4 Input file: speed.jpg.....	24
3.5 Input file: syifahadju1.jpg.....	26
3.6 Input file: syifahadju2.png.....	27
3.7 Input file: flower2.png.....	29
BAGIAN III.....	31
HASIL ANALISIS PERCOBAAN ALGORITMA.....	31
BAGIAN IV.....	32
IMPLEMENTASI BONUS.....	32
LINK REPOSITORY.....	34
TABEL CHECKLIST.....	34

BAGIAN I

ALGORITMA DIVIDE AND CONQUER

Algoritma *Divide and Conquer* merupakan salah satu strategi algoritma yang dapat digunakan sebagai teknik pemecahan masalah dari suatu persoalan. Algoritma ini dikenali dengan adanya pemecahan suatu persoalan menjadi beberapa upa-persoalan supaya lebih mudah diselesaikan. Algoritma ini terdiri dari tiga proses, yakni:

- *Divide*, membagi suatu persoalan menjadi upa-upa persoalan yang mirip tapi lebih kecil
- *Conquer*, menyelesaikan setiap upa-persoalan (dapat secara rekursif)
- *Combine*, menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Beberapa contoh persoalan yang dapat diselesaikan dengan algoritma *Divide and Conquer* adalah persoalan sorting yang dapat diselesaikan dengan *merge sort* atau *quick sort*, pencarian *closest pair*, dan Algoritma *Strassen*

Selain beberapa persoalan yang disebutkan sebelumnya, di sini algoritma Divide and Conquer difokuskan sebagai strategi yang dipakai dalam kompresi gambar dengan metode Quadtree. Cara kerja algoritma *Divide and Conquer* dalam persoalan ini adalah sebagai berikut:

1. Inisialisasi input-input yang diperlukan pada kompresi gambar, seperti alamat dari gambar, metode perhitungan error (variance, MAD, MPD, dan Entropy), *threshold*, dan *minimum block size*. Buatlah class Quadrant sebagai *node* yang menyimpan informasi dari *node* pada Quadtree beserta metode-metode *getter* atau metode yang diperlukan untuk proses kompresi. Sebelum dilakukan kompresi, gambar yang dikompresi diperoleh dengan membaca file dari alamat
2. Kompresi dimulai dengan mengambil blok gambar yang ukurannya sama dengan ukuran gambar. Blok dievaluasi berdasarkan ukuran blok dengan *minimum block size* yang ditentukan serta nilai perhitungan error dari blok berdasarkan metode perhitungan error yang dipilih. Blok akan dinyatakan tidak dapat dibagi menjadi blok-blok apabila memenuhi dua dari kondisi berikut:
 - Jika nilai hasil perhitungan error blok kurang dari nilai *threshold*
 - Jika ukuran blok awal kurang dari sama dengan dari *minimum block size* atau ukuran blok setelah dibagi menjadi empat kurang dari *minimum block size*

Apabila blok memenuhi salah satu dari kondisi berikut, *leaf* Quadrant (*node* daun) dari quadtree akan dibentuk dari blok yang selanjutnya dilakukan normalisasi warna dengan nilai *average* rgb blok. Hasil *leaf* quadrant yang telah dinormalisasi warnanya kemudian di-*return* dari metode compress.

3. Jika blok tidak memenuhi kedua syarat yang dijelaskan sebelumnya maka blok dinyatakan belum homogen sehingga blok gambar dapat dibagi menjadi empat blok gambar yang lebih kecil. Hal ini dilakukan dengan membuat empat Quadrant children baru yang kemudian tiap Quadrant *children* menyimpan hasil metode compression dengan parameter koordinat (parameter *x*, *y*), *width*, dan *height*-nya telah disesuaikan dengan masing-masing Quadrant (Kanan atas, kiri atas, kiri bawah, kanan bawah) yang total ukuran dari tiap *width* dan *height*-nya merupakan ukuran dari blok *parent* (membagi setara). Proses ini dilakukan secara rekursif dengan mengulangi langkah 2-3 yang dilakukan sebelumnya hingga blok tidak bisa dibagi kembali.
4. Pohon Quadtree kemudian dikonstruksi dari proses rekursif dengan membentuk Quadrant *parent* yang Quadrant *children-children* dari hasil penyimpanan kompresi dijadikan sebagai *children* dari Quadrant *parent* yang kemudian di-*return* dari method sebagai bentuk akhir dari Quadtree yang dibentuk.

BAGIAN II

SOURCE PROGRAM

Program utama ditulis dalam bahasa Java dengan memakai *packages* di antaranya:

1. Java.util
2. java.io
3. javax.imageio

4. Javax.awt

Program ini terdiri atas 3 *files* yang bersesuaian dengan kelas nya dengan satu *file* sebagai Main driver dari programnya. *Files* pada program ini adalah sebagai berikut

- Main.java
- Quadrant.java
- Quadtree.java

Kelas Main

```
src > J Main.java > Main > main(String[])
1 import java.util.Scanner;
2
3 public class Main {
4     Run | Debug
5     public static void main(String[] args) {
6         double threshold;
7         int errorMethod, minBlockSize;
8         Scanner scanner = new Scanner(System.in);
9         System.out.println("Enter the input image path (absolute path):");
10        System.out.print(s:>> " );
11        String inputPath = scanner.nextLine();
12
13        do {
14            System.out.println();
15            System.out.println("Enter the error method:");
16            System.out.println("0 Variance");
17            System.out.println("1 Mean Absolute Deviation");
18            System.out.println("2 Max Pixel Difference");
19            System.out.println("3 Entropy");
20            System.out.print(s:>> " );
21            errorMethod = Integer.parseInt(scanner.nextLine());
22            if (errorMethod < 0 || errorMethod > 4) {
23                System.out.println("Invalid error method. Please enter a number between 0 and 3.");
24                System.out.println();
25            } else {
26                break;
27            }
28        } while (true);
29
30        do {
31            System.out.println();
32            System.out.println("Enter the threshold for compression:");
33            System.out.print(s:>> " );
34            threshold = Double.parseDouble(scanner.nextLine());
35            if (threshold < 0) {
36                System.out.println("Invalid threshold number");
37            }
38        } while (true);
39
40        System.out.println();
41        System.out.println("Compression process started...");
42        System.out.println("Please wait until the process is completed.");
43        System.out.println("The compressed image will be saved to the specified path.");
44        System.out.println("Press any key to exit the program.");
45    }
46}
```

```

36         System.out.println();
37     } else {
38         if (errorMethod == 3) {
39             if (threshold > 8.0) {
40                 System.out.println(x:"Invalid threshold number for entropy. Please enter a n
41             } else {
42                 break;
43             }
44         } else if (errorMethod == 1 || errorMethod == 2) {
45             if (threshold > 255.0) {
46                 System.out.println(x:"Invalid threshold number for mean absolute deviation o
47             } else {
48                 break;
49             }
50         } else {
51             break;
52         }
53     }
54 } while (true);
55
56 do {
57     System.out.println();
58     System.out.println(x:"Enter the minimum block size:");
59     System.out.print(s:>> );
60     minBlockSize = Integer.parseInt(scanner.nextLine());
61     if (minBlockSize < 1) {
62         System.out.println(x:"Invalid minimum block size.");
63         System.out.println();
64     } else {
65         break;
66     }
67 } while (true);
68 System.out.println();

```

```

69     System.out.println(x:"Enter the output image path (absolute path):");
70     System.out.print(s:>> );
71     String outputPath = scanner.nextLine();
72     System.out.println();
73     System.out.println(x:"Enter the output GIF path (absolute path):");
74     System.out.print(s:>> );
75     String GIFPath = scanner.nextLine();
76
77
78     QuadtreeCompressor compressor = new QuadTreeCompressor(inputPath, errorMethod, threshold, minBlockSize, outputPath, GIFPath);
79     compressor.startCompress();
80     scanner.close();
81 }
82 }

```

Main.java

Kelas Quadrant

```
src > J Quadrant.java > ...
1
2 import java.awt.*;
3
4 public class Quadrant {
5     private int x, y, height, width;
6     private color color;
7     private Quadrant[] children;
8     private boolean isLeaf;
9
10    public Quadrant(int x, int y, int width, int height) {
11        this.x = x;
12        this.y = y;
13        this.width = width;
14        this.height = height;
15        this.isLeaf = false;
16        this.children = null;
17    }
18
19    public int getX() { return x; }
20    public int getY() { return y; }
21    public int getWidth() { return width; }
22    public int getHeight() { return height; }
23    public int getNodesCount(Quadrant node) {
24        if (node.isLeaf) return 1;
25        int count = 1;
26        if (node.getChildren() != null) {
27            for (Quadrant child : node.getChildren()) {
28                count += getNodesCount(child);
29            }
30        }
31        return count;
32    }
33    public int getMaxDepth(Quadrant node) {
34        if (node.isLeaf) return 1;
35        int maxDepth = 0;
```

```
36     if (node.getChildren() != null) {
37         for (Quadrant child : node.getChildren()) {
38             maxDepth = Math.max(maxDepth, getMaxDepth(child));
39         }
40     }
41     return maxDepth + 1;
42 }
43 public Color getColor() { return color; }
44 public Quadrant[] getChildren() { return children; }
45 public boolean isLeaf() { return isLeaf; }
46
47 public void setColor(Color color) { this.color = color; }
48 public void setChildren(Quadrant[] children) { this.children = children; }
49 public void setLeaf(boolean isLeaf) { this.isLeaf = isLeaf; }
50 }
51
```

Quadrant.java

Kelas Quadtree

```
src > J QuadTreeCompressor.java > QuadTreeCompressor > startCompress()
1 import com.madgag.gif.fmsware.AnimatedGifEncoder;
2 import java.awt.*;
3 import java.awt.image.BufferedImage;
4 import java.io.File;
5 import java.io.IOException;
6 import java.util.Iterator;
7 import javax.imageio.IIOImage;
8 import javax.imageio.ImageIO;
9 import javax.imageio.ImageWriteParam;
10 import javax.imageio.ImageWriter;
11 import javax.imageio.stream.FileImageOutputStream;
12 import javax.swing.*;
13
14
15 public class QuadTreeCompressor {
16     private String inputPath;
17     private int errorMethod;
18     private double threshold;
19     private int minBlockSize;
20     private String outputPath;
21     private String GIFPath;
22
23     public QuadTreeCompressor(String inputPath, int errorMethod, double threshold,
24                               int minBlockSize, String outputPath, String GIFPath) {
25         this.inputPath = inputPath;
26         this.errorMethod = errorMethod;
27         this.threshold = threshold;
28         this.minBlockSize = minBlockSize;
29         this.outputPath = outputPath;
30         this.GIFPath = GIFPath;
31     }
32
33     public void startCompress(){
34         try{
35             // Load gambar
36             BufferedImage originalImage = ImageIO.read(new File(inputPath));
```

```
37
38     File originalFile = new File(inputPath);
39     System.out.println("Loaded image: " + outputPath);
40
41     // Kompresi gambar
42     System.out.println("Starting compression...");
43     int width = originalImage.getWidth();
44     int height = originalImage.getHeight();
45     Long mulai = System.currentTimeMillis();
46     Quadrant compressed = compress(originalImage, x:0, y:0, width, height);
47     BufferedImage compressedImage = imageReconstruction(compressed);
48     Long selesai = System.currentTimeMillis();
49
50     // Menyimpan ke memori (bukan ke file fisik)
51
52     File compressedFile = new File(outputPath);
53     Iterator<ImageWriter> writers = ImageIO.getImageWritersByFormatName(formatName:"jpg");
54     if (!writers.hasNext()) {
55         throw new IllegalStateException(s:"No writers found for JPEG format.");
56     }
57     ImageWriter writer = writers.next();
58
59     ImageWriteParam param = writer.getDefaultWriteParam();
60     param.setCompressionMode(ImageWriteParam.MODE_EXPLICIT);
61     param.setCompressionQuality(quality:0.5f);
62
63     try (FileImageOutputStream output = new FileImageOutputStream(compressedFile)) {
64         writer.setOutput(output);
65         writer.write(streamMetadata:null, new IIOImage(compressedImage, thumbnails:null, metadata:null), param);
66         writer.dispose();
67     }
68     System.out.println("Image saved to: " + outputPath + "\n");
69
70     generateGIF(compressed, originalImage.getWidth(), originalImage.getHeight(), GIFPath);
```

```
71     System.out.println("GIF saved to: " + GIFPath);
72
73     // Ukuran dalam memori setelah kompresi
74     long compressedFileSize = compressedFile.length();
75     long originalFileSize = originalFile.length(); // ukuran file asli dalam byte
76     long compressionRatio = (long) ((1 - (double) compressedFileSize / originalFileSize) * 100);
77
78     // Menampilkan informasi kompresi
79     System.out.println("Compression completed in " + (selesai - mulai) + " ms");
80     System.out.println("Original image size: " + originalFileSize + " bytes");
81     System.out.println("Compressed image size: " + compressedFileSize + " bytes");
82     System.out.println("Percentase kompresi: " + compressionRatio + "%");
83     System.out.println("Kedalaman pohon: " + compressed.getMaxDepth(compressed));
84     System.out.println("Banyak simpul: " + compressed.getNodesCount(compressed));
85
86     // Menampilkan gambar terkompresi
87     displayImage(compressedImage);
88
89
90 } catch (IOException e) {
91     System.err.println("Error loading the image: " + e.getMessage());
92     e.printStackTrace();
93 }
94
95
96 // Method compress
97 private Quadrant compress(BufferedImage image, int x, int y, int width, int height) {
98     int w1 = width / 2;
99     int w2 = width - w1;
100    int h1 = height / 2;
101    int h2 = height - h1;
102    int totalPixel = width * height;
103    int totalSubPixel1 = w1 * h1;
```

```
105     int totalSubPixel2 = w2 * h1;
106     int totalSubPixel3 = w1 * h2;
107     int totalSubPixel4 = w2 * h2;
108     if ((totalPixel >= minBlockSize && (totalSubPixel1 < minBlockSize &&
109         totalSubPixel2 < minBlockSize && totalSubPixel3 < minBlockSize &&
110         totalSubPixel4 < minBlockSize) || totalPixel <= minBlockSize)) {
111         Quadrant leaf = new Quadrant(x, y, width, height);
112         leaf.setColor(averageColor(image, x, y, width, height));
113         leaf.setLeaf(isLeaf:true);
114         return leaf;
115     }
116
117     double error = ErrorMeasurement(image, x, y, width, height);
118     // System.out.printf("Error: %.2f, Threshold: %.2f\n", error, threshold);
119     if (error < threshold) {
120         Quadrant leaf = new Quadrant(x, y, width, height);
121         leaf.setColor(averageColor(image, x, y, width, height));
122         leaf.setLeaf(isLeaf:true);
123         return leaf;
124     }
125
126     // Rekursi dengan bagi gambar menjadi 4 kuadran
127     Quadrant[] children = new Quadrant[4];
128     children[0] = compress(image, x, y, w1, h1);
129     children[1] = compress(image, x + w1, y, w2, h1);
130     children[2] = compress(image, x, y + h1, w1, h2);
131     children[3] = compress(image, x + w1, y + h1, w2, h2);
132
133     Quadrant parent = new Quadrant(x, y, width, height);
134     parent.setChildren(children);
135     parent.setLeaf(isLeaf:false);
136
137     Color parentColor = averageColorFromChildren(parent); //set warna buat node yang bukan leaf
138     parent.setColor(parentColor);
```

```
139     return parent;
140 }
141 public BufferedImage imageReconstruction(Quadrant root) {
142     BufferedImage reconstructedImage = new BufferedImage(root.getWidth(), root.getHeight(), BufferedImage.TYPE_INT_RGB);
143     imageReconstructionProcess(root, reconstructedImage);
144     return reconstructedImage;
145 }
146
147 // Method reconstruct gambar hasil compress
148 public void imageReconstructionProcess(Quadrant node, BufferedImage reconstructedImage) {
149     if(node.isleaf()){
150         for (int i= node.getY();i<node.getY()+node.getHeight();i++){
151             for (int j=node.getX();j<node.getX()+node.getWidth();j++){
152                 reconstructedImage.setRGB(j,i,node.getColor().getRGB());
153             }
154         }
155     }else{
156         for (Quadrant child:node.getChildren()){
157             imageReconstructionProcess(child, reconstructedImage);
158         }
159     }
160 }
161
162 // Method untuk menampilkan gambar
163 public static void displayImage(BufferedImage image) {
164     System.out.println("Displaying image...");
165
166     Dimension screenSize = Toolkit.getDefaultToolkit().getScreenSize();
167     int maxWidth = screenSize.width - 100;
168     int maxHeight = screenSize.height - 100;
169
170     int imgWidth = image.getWidth();
171     int imgHeight = image.getHeight();
172 }
```

```
173     BufferedImage displayImage = image;
174
175     // Menyesuaikan ukuran gambar jika lebih besar dari layar
176     if (imgWidth > maxWidth || imgHeight > maxHeight) {
177         double widthRatio = (double) maxWidth / imgWidth;
178         double heightRatio = (double) maxHeight / imgHeight;
179         double scale = Math.min(widthRatio, heightRatio);
180
181         int newWidth = (int) (imgWidth * scale);
182         int newHeight = (int) (imgHeight * scale);
183
184         Image scaledImg = image.getScaledInstance(newWidth, newHeight, Image.SCALE_SMOOTH);
185
186         // Konversi scaled image ke BufferedImage
187         displayImage = new BufferedImage(newWidth, newHeight, BufferedImage.TYPE_INT_RGB);
188         Graphics2D g2d = displayImage.createGraphics();
189         g2d.drawImage(scaledImg, x:0, y:0, observer:null);
190         g2d.dispose();
191     }
192
193     // Menampilkan gambar
194     JFrame frame = new JFrame(title:"Image Viewer");
195     JLabel label = new JLabel(new ImageIcon(displayImage));
196     frame.getContentPane().add(label, BorderLayout.CENTER);
197     frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
198     frame.pack();
199     frame.setLocationRelativeTo(c:null);
200     frame.setVisible(b:true);
201 }
202
203     private void generateGIF(Quadrant root, int width, int height, String GIFpath) {
204     try {
205         AnimatedGifEncoder encoder = new AnimatedGifEncoder();
206         encoder.setRepeat(iter:0);
```

```
207     encoder.setDelay(ms:500);
208     encoder.start(GIFpath);
209
210     int maxDepth = root.getMaxDepth(root);
211     for (int i = 0; i <= maxDepth; i++) { //buat frame
212         BufferedImage frame = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
213         drawFrame(root, frame, i);
214         encoder.addFrame(frame);
215     }
216
217     BufferedImage finalFrame = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
218     imageReconstructionProcess(root, finalFrame);
219     encoder.addFrame(finalFrame);
220
221     encoder.finish();
222 } catch (Exception e) {
223     System.err.println("Error generating GIF: " + e.getMessage());
224 }
225
226 }
227
228 private void drawFrame(Quadrant node, BufferedImage frame, int targetDepth) {
229     drawFrameRecursive(node, frame, targetDepth, currentDepth:0);
230 }
231
232 private void drawFrameRecursive(Quadrant node, BufferedImage frame, int targetDepth, int currentDepth) {
233     if (node == null) return;
234
235     if (node.isLeaf() || currentDepth == targetDepth) {
236         for (int i = node.getY(); i < node.getY() + node.getHeight(); i++) {
237             for (int j = node.getX(); j < node.getX() + node.getWidth(); j++) {
238                 frame.setRGB(j, i, node.getColor().getRGB());
239             }
240         }
241     }
242 }
```

```
241     } else {
242         for (Quadrant child : node.getChildren()) {
243             drawFrameRecursive(child, frame, targetDepth, currentDepth + 1);
244         }
245     }
246 }
247
248
249
250 // Method error Measurement
251 private double ErrorMeasurement(BufferedImage image, int x, int y, int width, int height) {
252     return switch(this.errorMethod) {
253         case 0 -> varianceMeasurement(image, x, y, width, height);
254         case 1 -> MADMeasurement(image, x, y, width, height);
255         case 2 -> maxPixelDifferenceMeasurement(image, x, y, width, height);
256         case 3 -> entropyMeasurement(image, x, y, width, height);
257         default -> throw new IllegalArgumentException("Error measurement method invalid: " + this.errorMethod);
258     };
259 }
260
261 // Variance
262 private double varianceMeasurement(BufferedImage image, int x, int y, int width, int height) {
263     double varianceR = calculateVariance(image, x, y, width, height, channel:'R');
264     double varianceG = calculateVariance(image, x, y, width, height, channel:'G');
265     double varianceB = calculateVariance(image, x, y, width, height, channel:'B');
266     return (varianceR + varianceG + varianceB) / 3.0;
267 }
268
269 private double calculateVariance(BufferedImage image, int x, int y, int width, int height, char channel) {
270     double variance = 0.0;
271     Color avgColor = averageColor(image, x, y, width, height);
272     for (int i = y; i < y + height; i++) {
273         for (int j = x; j < x + width; j++) {
274             Color pixelColor = new Color(image.getRGB(j, i));
```

```

275         if (channel == 'R') {
276             variance += Math.pow(pixelColor.getRed() - avgColor.getRed(), b:2);
277         } else if (channel == 'G') {
278             variance += Math.pow(pixelColor.getGreen() - avgColor.getGreen(), b:2);
279         } else if (channel == 'B') {
280             variance += Math.pow(pixelColor.getBlue() - avgColor.getBlue(), b:2);
281         }
282     }
283     return variance / (width * height);
284 }
285
286
287
288 // Mean Absolute Deviation
289 private double MADMeasurement(BufferedImage image, int x, int y, int width, int height) {
290     int pixelCount = width * height;
291     Color avgColor = averageColor(image, x, y, width, height);
292     double MADR = 0, MADG = 0, MADB = 0;
293
294     for (int i = y; i < y + height; i++) {
295         for (int j = x; j < x + width; j++) {
296             Color pixelColor = new Color(image.getRGB(j, i));
297             MADR += Math.abs(pixelColor.getRed() - avgColor.getRed());
298             MADG += Math.abs(pixelColor.getGreen() - avgColor.getGreen());
299             MADB += Math.abs(pixelColor.getBlue() - avgColor.getBlue());
300         }
301     }
302     return (MADR + MADG + MADB) / (pixelCount * 3.0);
303 }
304
305 // Max Pixel Difference
306 private double maxPixelDifferenceMeasurement(BufferedImage image, int x, int y, int width, int height) {
307     int minR = 255, minG = 255, minB = 255;
308     int maxR = 0, maxG = 0, maxB = 0;

```

```

310     for (int i = y; i < y + height; i++) {
311         for (int j = x; j < x + width; j++) {
312             Color pixelColor = new Color(image.getRGB(j, i));
313
314             minR = Math.min(minR, pixelColor.getRed());
315             minG = Math.min(minG, pixelColor.getGreen());
316             minB = Math.min(minB, pixelColor.getBlue());
317             maxR = Math.max(maxR, pixelColor.getRed());
318             maxG = Math.max(maxG, pixelColor.getGreen());
319             maxB = Math.max(maxB, pixelColor.getBlue());
320         }
321     }
322
323     int diffR = maxR - minR;
324     int diffG = maxG - minG;
325     int diffB = maxB - minB;
326     return (diffR + diffG + diffB) / 3.0;
327 }
328
329 // Entropy
330 private double entropyMeasurement(BufferedImage image, int x, int y, int width, int height) {
331     double entropyR = channelEntropy(image, x, y, width, height, channel:'R');
332     double entropyG = channelEntropy(image, x, y, width, height, channel:'G');
333     double entropyB = channelEntropy(image, x, y, width, height, channel:'B');
334     return (entropyR + entropyG + entropyB) / 3.0;
335 }
336
337 private double channelEntropy(BufferedImage image, int x, int y, int width, int height, char channel) {
338     int[] colorfreqArray = new int[256];
339     int totalPixels = width * height;
340
341     for (int i = y; i < y + height; i++) {
342         for (int j = x; j < x + width; j++) {
343             Color pixelColor = new Color(image.getRGB(j, i));

```

```
344     int value = 0;
345     if (channel == 'R') {
346         value = pixelColor.getRed();
347     } else if (channel == 'G') {
348         value = pixelColor.getGreen();
349     } else if (channel == 'B') {
350         value = pixelColor.getBlue();
351     }
352     colorfreqArray[value]++;
353 }
354
355
356     double channelEntropy = 0.0;
357     for (int i = 0; i < colorfreqArray.length; i++) {
358         int colorFreq = colorfreqArray[i];
359         double colorProbability = (double) colorFreq / totalPixels;
360         if (colorProbability > 0) {
361             channelEntropy -= colorProbability * Math.log(colorProbability) / Math.log(a:2);
362         }
363     }
364     return channelEntropy;
365 }
366
367 private Color averageColor(BufferedImage image, int x, int y, int width, int height) {
368     long r = 0, g = 0, b = 0;
369     int pixelCount = width * height;
370     for (int i = y; i < y + height; i++) {
371         for (int j = x; j < x + width; j++) {
372             Color pixelColor = new Color(image.getRGB(j, i));
373             r += pixelColor.getRed();
374             g += pixelColor.getGreen();
375             b += pixelColor.getBlue();
376         }
377     }
378 }
```

```
378     |     return new Color((int)(r / pixelCount), (int)(g / pixelCount), (int)(b / pixelCount));
379   }
380
381     private Color averageColorFromChildren(Quadrant node) {
382       if (node.getChildren() == null || node.getChildren().length != 4) {
383         throw new IllegalArgumentException("Node does not have 4 children");
384       }
385
386       Long r = 0, g = 0, b = 0;
387       for (Quadrant child : node.getChildren()) {
388         Color c = child.getColor();
389         r += c.getRed();
390         g += c.getGreen();
391         b += c.getBlue();
392       }
393
394       int avgR = (int)(r / 4);
395       int avgG = (int)(g / 4);
396       int avgB = (int)(b / 4);
397
398       return new Color(avgR, avgG, avgB);
399     }
400   }
401 }
```

QuadTreeCompressor.java

BAGIAN III

SCREENSHOT HASIL TEST

3.1 Input file: flower.jpg



Hasil :



3.2 Input file: trump.jpg



Hasil :

```
Enter the input image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trump.jpg

Enter the error method:
0 Variance
1 Mean Absolute Deviation
2 Max Pixel Difference
3 Entropy
>> 1

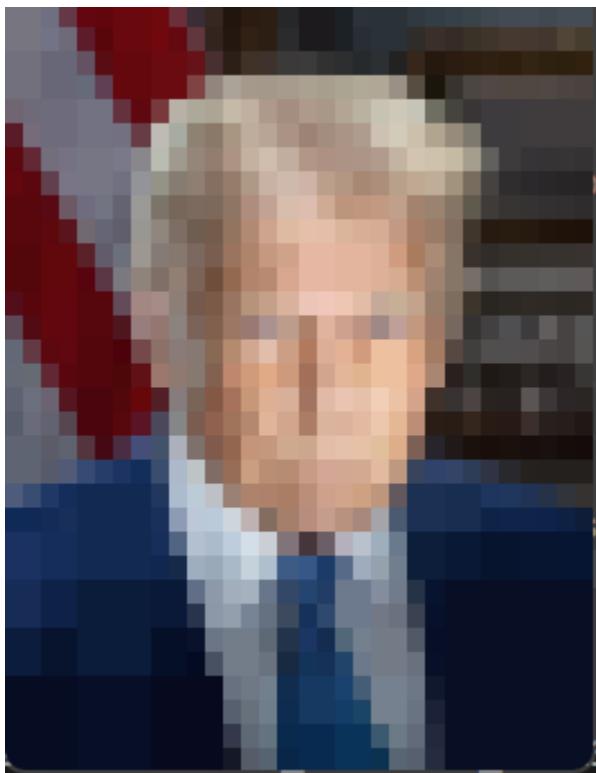
Enter the threshold for compression:
>> 8

Enter the minimum block size:
>> 32

Enter the output image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trumps.jpg

Enter the output GIF path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trumps.gif
Loaded image: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trump.jpg
Starting compression...
Image saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trumps.jpg

GIF saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\trumps.gif
Compression completed in 84 ms
Original image size: 6964 bytes
Compressed image size: 5436 bytes
Percentase kompresi: 21%
Kedalaman pohon: 6
Banyak simpul: 993
Displaying image...
```



3.3 Input file: messi.jpg



Hasil:

```
Enter the input image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messi.jpg

Enter the error method:
0 Variance
1 Mean Absolute Deviation
2 Max Pixel Difference
3 Entropy
>> 2

Enter the threshold for compression:
>> 12

Enter the minimum block size:
>> 12

Enter the output image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messiu.jpg

Enter the output GIF path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messiu.gif
Loaded image: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messi.jpg
Starting compression...
Image saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messiu.jpg

GIF saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\messiu.gif
Compression completed in 70 ms
Original image size: 10484 bytes
Compressed image size: 7566 bytes
Percentase kompresi: 27%
Kedalaman pohon: 7
Banyak simpul: 5305
Displaying image...
```



3.4 Input file: speed.jpg



Hasil :

```
Enter the input image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speed.jpg

Enter the error method:
0 Variance
1 Mean Absolute Deviation
2 Max Pixel Difference
3 Entropy
>> 3

Enter the threshold for compression:
>> 3

Enter the minimum block size:
>> 9

Enter the output image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speedy.jpg

Enter the output GIF path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speedy.gif
Loaded image: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speed.jpg
Starting compression...
Image saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speedy.jpg

GIF saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\speedy.gif
Compression completed in 84 ms
Original image size: 6399 bytes
Compressed image size: 5624 bytes
Persentase kompresi: 12%
Kedalaman pohon: 7
Banyak simpul: 4473
Displaying image...
```



3.5 Input file: syifahadju1.jpg



Hasil :

```
Enter the input image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju1.png

Enter the error method:
0 Variance
1 Mean Absolute Deviation
2 Max Pixel Difference
3 Entropy
>> 0

Enter the threshold for compression:
>> 64

Enter the minimum block size:
>> 8

Enter the output image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju.jpg

Enter the output GIF path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju.gif
Loaded image: D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju1.png
Starting compression...
Image saved to: D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju.jpg

GIF saved to: D:\Codes\SEMESTER4IF\Codes\Jawa\Tucil2_13523023_13523025\syifahadju.gif
Compression completed in 96 ms
Original image size: 83780 bytes
Compressed image size: 6296 bytes
Percentase kompresi: 92%
Kedalaman pohon: 7
Banyak simpul: 4517
Displaying image...
```



3.6 Input file: syifahadju2.png



Hasil :

```
Enter the error method:  
0 Variance  
1 Mean Absolute Deviation  
2 Max Pixel Difference  
3 Entropy  
>> 2  
  
Enter the threshold for compression:  
>> 14  
  
Enter the minimum block size:  
>> 10  
  
Enter the output image path (absolute path):  
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\syifahadju2.jpg  
  
Enter the output GIF path (absolute path):  
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\syifahadju2.gif  
Loaded image: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\syifahadju2.png  
Starting compression...  
Image saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\syifahadju2.jpg  
  
GIF saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\syifahadju2.gif  
Compression completed in 116 ms  
Original image size: 332505 bytes  
Compressed image size: 18851 bytes  
Percentase kompresi: 94%  
Kedalaman pohon: 8  
Banyak simpul: 14893  
Displaying image...
```



3.7 Input file: flower2.png



Hasil :

```
Enter the input image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.png

Enter the error method:
0 Variance
1 Mean Absolute Deviation
2 Max Pixel Difference
3 Entropy
>> 3

Enter the threshold for compression:
>> 2

Enter the minimum block size:
>> 10

Enter the output image path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.jpg

Enter the output GIF path (absolute path):
>> D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.gif
Loaded image: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.png
Starting compression...
Image saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.jpg

GIF saved to: D:\Codes\SEMESTER4IF\Codes\Java\Tucil2_13523023_13523025\flower2.gif
Compression completed in 677 ms
Original image size: 1295659 bytes
Compressed image size: 83834 bytes
Persentase kompresi: 93%
Kedalaman pohon: 10
Banyak simpul: 29541
Displaying image...
```



BAGIAN III

HASIL ANALISIS PERCOBAAN

ALGORITMA

Berdasarkan percobaan yang dilakukan pada program kompresi gambar dengan Quadtree yang mengimplementasikan strategi algoritma *divide and conquer*, terbukti bahwa gambar dapat dikompresi secara baik. Kemampuan utama algoritma ini adalah untuk mengadaptasi proses kompresi berdasarkan kompleksitas lokal pada gambar. Area-area gambar yang homogen akan direpresentasikan dengan blok besar, sedangkan area yang lebih kompleks akan dipecah menjadi blok-blok yang lebih kecil. Proses kompresi gambar ini memiliki tiga operasi utama yaitu perhitungan rata-rata warna, perhitungan nilai error, dan rekonstruksi gambar. Fungsi perhitungan warna akan mengakses semua piksel dalam blok. Gambar dibagi menjadi blok-blok yang tidak saling tumpang tindih dan semua piksel hanya perlu diakses sekali maka kompleksitas untuk menghitung rata-rata warna seluruh gambar adalah $O(n)$.

Perhitungan nilai error dilakukan untuk setiap blok yang diakses saat proses rekursif. Operasi ini akan mengakses seluruh piksel dalam blok untuk menghitung error berdasarkan metode yang dipilih. Oleh karena itu total kompleksitas perhitungan nilai error adalah $O(n)$. Operasi rekonstruksi gambar juga dilakukan dengan cara demikian sehingga kompleksitasnya $O(n)$. Pada setiap rekursi, gambar dibagi menjadi empat kuadran, dan proses kompresi dapat dibagi menjadi dua yaitu saat n merupakan ukuran terkecil dan saat n lebih dari ukuran terkecil, dengan kasus ketika bukan ukuran terkecil akan dilakukan combine sebanyak jumlah piksel, sehingga dapat dituliskan :

$$T(n) = \begin{cases} 3n & \text{if } n \leq n_0 \\ 4T\left(\frac{n}{4}\right) + 3n + n & \text{if } n > n_0 \end{cases}$$

Dari teorema master, $T(n) = aT(n/b) + cn^d$, maka $a = 4$, $b = 4$, $c = 4$, dan $d = 1$, sehingga memenuhi *case 2* yaitu $a = b^d$. Proses kompresi gambar memiliki kompleksitas waktu sebagai berikut:

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

Kompleksitas ini cukup efisien untuk digunakan pada gambar berukuran sedang hingga besar karena hasil akhir dari kompresi menghasilkan representasi yang lebih ringan dalam hal jumlah data yang disimpan. Dengan demikian, algoritma kompresi berbasis *divide and conquer* ini menawarkan pendekatan yang fleksibel, adaptif, dan terstruktur untuk menyederhanakan informasi dua dimensi yang memerlukan pengendalian kualitas visual sekaligus penghematan penyimpanan.

BAGIAN IV

IMPLEMENTASI BONUS

Bonus yang diimplementasikan pada program kompresi gambar dengan Quadtree adalah visualisasi proses pembentukan Quadtree dalam kompresi gambar dengan format GIF. Fitur ini menggunakan *library* java yang terdapat pada <https://github.com/rtyley/animated-gif-lib-for-java>, yakni dengan “import com.madgag.gif.fmsware.AnimatedGifEncoder”. Dengan *library* ini, program dapat menghasilkan file .gif dari kumpulan *frame* yang memperlihatkan proses pembentukan pohon Quadtree secara bertahap.

Mekanisme GIF pada program bekerja yakni setelah proses kompresi selesai, setiap tahapan pembentukan pohon Quadtree diubah menjadi gambar (*frame*) berdasarkan kedalaman (*depth*) Quadrant (*node*) saat itu. Setiap *frame* menunjukkan hasil rekonstruksi gambar hingga kedalaman tertentu, yang menunjukkan proses bagaimana pohon Quadtree bertambah kompleks secara visual. Pada metode generateGIF() pada Quadtree.java, dilakukan inisialisasi *encoder* GIF, penyusunan *frame* dari kedalaman 0 sampai kedalaman maksimal pohon Quadtree, penambahan *frame* akhir berupa gambar hasil rekonstruksi penuh, dan penyimpanan GIF ke *path* yang sudah ditentukan (GIFPath).

Berikut ini adalah *snapshot code* yang berkaitan dengan proses implementasi bonus GIF pada program:

```
private void generateGIF(Quadrant root, int width, int height, String GIFpath) {
    try {
        AnimatedGifEncoder encoder = new AnimatedGifEncoder();
        encoder.setRepeat(iter:0);
        encoder.setDelay(ms:500);
        encoder.start(GIFpath);

        int maxDepth = root.getMaxDepth(root);
        for (int i = 0; i <= maxDepth; i++) { //buat frame
            BufferedImage frame = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
            drawFrame(root, frame, i);
            encoder.addFrame(frame);
        }

        BufferedImage finalFrame = new BufferedImage(width, height, BufferedImage.TYPE_INT_RGB);
        imageReconstructionProcess(root, finalFrame);
        encoder.addFrame(finalFrame);

        encoder.finish();
    }
}
```

```
        } catch (Exception e) {
            System.err.println("Error generating GIF: " + e.getMessage());
        }
    }

private void drawFrame(Quadrant node, BufferedImage frame, int targetDepth) {
    drawFrameRecursive(node, frame, targetDepth, currentDepth:0);
}

private void drawFrameRecursive(Quadrant node, BufferedImage frame, int targetDepth, int currentDepth) {
    if (node == null) return;

    if (node.isLeaf() || currentDepth == targetDepth) {
        for (int i = node.getY(); i < node.getY() + node.getHeight(); i++) {
            for (int j = node.getX(); j < node.getX() + node.getWidth(); j++) {
                frame.setRGB(j, i, node.getColor().getRGB());
            }
        }
    } else {
        for (Quadrant child : node.getChildren()) {
            drawFrameRecursive(child, frame, targetDepth, currentDepth + 1);
        }
    }
}
```

AnimatedGifEncoder.java

LINK REPOSITORY

https://github.com/AgungLucker/Tucil2_13523023_13523025

TABEL CHECKLIST

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