# LAPORAN TUGAS KECIL II
# IF2211 STRATEGI ALGORITMA

# QUADTREE IMAGE COMPRESSION: DIVIDE & CONQUER APPROACH

**Disusun oleh:**

Nathan Jovial Hartono - 13523032


**PROGRAM STUDI TEKNIK INFORMATIKA**

**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**

**INSTITUT TEKNOLOGI BANDUNG**

**2024**

# TABLE OF CONTENTS

# BAB I
# DESKRIPSI MASALAH

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

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

# BAB II
# IMPLEMENTASI ALGORITMA

Pemanfaatan algoritma divide and conquer quadtree pada image compression sangat mirip dengan penggunaan quadtree pada umumnya. Quadtree, walaupun merupakan struktur data dengan empat node child, merupakan metode penyimpanan data dalam approach divide and conquer sebuah objek berdimensi dua, seperti sebuah image. Contoh paling umum penggunaan quadtree yaitu pengelompokkan titik pada sebuah bidang persegi. Data lokasi titik diuji dengan sebuah threshold atau kondisi yang memeriksa apakah bidang tersebut harus dibagi menjadi empat bagian berbeda lagi atau tidak. Hal yang sama dapat diimplementasikan pada image compression, tetapi tidak memanfaatkan posisi sebuah titik melainkan menghitung rata - rata RGB value sebuah block dan membandingkannya dengan sebuah threshold, dengan catatan threshold yang digunakan pada tugas ini sudah disediakan rumusnya.

Berikut langkah - langkah algoritma divide and conquer:

1. Tentukan posisi awal x,y dan akhir x,y pada gambar. Gambar tidak harus memiliki ukuran n x n. Saat pertama kali gambar diproses, awal dan akhir x,y merupakan dimensi original gambar.
2. Lakukan perhitungan error dan rata - rata setiap channel warna pada daerah gambar tersebut. Terdapat lima metode perhitungan error yang dapat digunakan: variance, mean absolute deviation, max pixel difference, entropy, dan structural similarity index (SSIM).
3. Bandingkan perhitungan error dengan threshold berkorespondensi dengan metode perhitungan error yang dipilih. Note: threshold ditentukan oleh user pada awal berjalan program
4. Apabila nilai error lebih besar dari threshold yang ditentukan dan luas daerah lebih besar dari minimum block size yang ditentukan, lakukan pembagian daerah menjadi 4 sub daerah dengan posisi awal dan akhir x,y masing - masing daerah, pembagian daerah ini symmetrical sehingga luas daerah yang terbagi sama. Ulangi langkah pertama. Note: minimum block size ditentukan oleh user pada awal berjalan program.
5. Apabila nilai error kurang dari threshold yang ditentukan atau luas daerah lebih kecil dari minimum block size yang ditentukan, maka proses berhenti dan daerah ini disimpan sebagai node-leaf dari Quadtree. Note: recursive base case.

Rekonstruksi ulang gambar melakukan traversal pada node secara DFS untuk mendapatkan informasi rata - rata pixel pada daerah gambar tersebut. Dengan metode ini, tidak akan ada informasi yang saling overwrite dan hanya tersimpan pada daerah masing - masing.

# BAB III
# SOURCE CODE PROGRAM

## I. ImageProcessing.java

```
package QuadTree;

import java.awt.Color;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.Scanner;
import javax.imageio.ImageIO;
import javax.swing.ImageIcon;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JScrollPane;

public class ImageProcessing {

	private BufferedImage image;
	private int width;
	private int height;
	private String absPath;
	private String extension;
	private double fileSize;

	// Default constructor
	public ImageProcessing() {
	this.image = null;
	this.absPath = " ";
	this.width = 0;
	this.height = 0;
	this.extension = " ";
	this.fileSize = 0;
	}

	// User-defined constructor with absPath
	public ImageProcessing(String absPath) throws IOException {
	this.absPath = absPath;
	this.image = ImageIO.read(new File(absPath));
```

```
      this.width = this.image.getWidth();
      this.height = this.image.getHeight();
      this.extension = getFileExtension(absPath);
      this.fileSize = new File(absPath).length();
      }

      // Loads image not from constructor
      public void loadImage(String absPath) throws IOException {
      this.absPath = absPath;
      this.image = ImageIO.read(new File(absPath));
      this.width = this.image.getWidth();
      this.height = this.image.getHeight();
      this.extension = getFileExtension(absPath);
      this.fileSize = new File(absPath).length();
      }

      // Method for user input absolute path
      public void inputAbsPath(){

      Scanner scanner = new Scanner(System.in);
      System.out.print("Enter the absolute path of the image: ");
      this.absPath = scanner.nextLine();


      try{
            loadImage(this.absPath);
            System.out.println("Image loaded succesfully.");
      }
      catch(IOException e){
            System.out.println("Error loading image. Check the file
path.");
      }
      }

      // Returns width attribute
      public int getWidth(){
      return this.width;
      }

      // Returns height attribute
      public int getHeight(){
      return this.height;
      }
```

```
	// Returns extension
	public String getExtension(){
	return this.extension;
	}

	// Returns image buffer
	public BufferedImage getImage(){
	return this.image;
	}

	// Returns image file size
	public double getFileSize(){
	return this.fileSize;
	}

	// Returns file absolute path
	public String getAbsPath(){
	return this.absPath;
	}

	// Returns true if image is null
	public Boolean isImageEmpty(){
	return this.image == null;
	}

	// Returns a Pixel object at (x,y) coordinate
	public Pixel getPixelValue(int x, int y) {
	if (x < 0 || x >= width || y < 0 || y >= height) {
		throw new IllegalArgumentException("Coordinates out of
bounds.");
	}
	int pixel = image.getRGB(x, y);
	Color color = new Color(pixel, true); // Extract color components
	return new Pixel(color.getRed(), color.getGreen(), color.getBlue());
	}

	// Returns file extension
	public static String getFileExtension(String fileName) {
	int lastDotIndex = fileName.lastIndexOf('.');
	if (lastDotIndex == -1 || lastDotIndex == fileName.length() - 1) {
		return ""; // No extension found
	}
```

```
        return fileName.substring(lastDotIndex + 1);
        }

        // Image viewer with JFrame
        public void viewImage() {
        if (isImageEmpty()) {
              System.out.println("No image loaded.");
              return;
        }

        JFrame frame = new JFrame("Image Viewer");
        frame.setDefaultCloseOperation(JFrame.DISPOSE_ON_CLOSE);
        frame.setSize(this.width, this.height);

        JLabel label = new JLabel(new ImageIcon(this.image));
        frame.add(new JScrollPane(label));

        frame.pack(); // Adjusts the frame size based on image size
        frame.setLocationRelativeTo(null); // Centers the frame
        frame.setVisible(true);
        }
}
```

## II. Pixel.java

```
package QuadTree;

public class Pixel {

        private int r;
        private int g;
        private int b;

        // Default constructor
        public Pixel(){
        this.r = 0;
        this.g = 0;
        this.b = 0;
        }

        // User-defined constructor with absPath
        public Pixel(int r, int g, int b){
```

```
this.r = r;
this.g = g;
this.b = b;
}

// Copy constructor
public Pixel(Pixel other){
this(other.r,other.g,other.b);
}

// Returns 'r' channel value of pixel
public int getR(){
return this.r;
}

// Returns 'g' channel value of pixel
public int getG(){
return this.g;
}

// Returns 'b' channel value of pixel
public int getB(){
return this.b;
}

// Sets the r, g, and b attributes of Pixel
public void setRGB(int r, int g, int b){
this.r = r;
this.b = b;
this.g = g;
}

// Adds pixel with other pixels
public void add(Pixel other){
this.r+=other.r;
this.b+=other.b;
this.g+=other.g;
}

// Substracts pixel with other pixels
public void minus(Pixel other){
this.r-=other.r;
this.b-=other.b;
```

```
        this.g-=other.g;
        }

        // Mean division of pixel from n pixels
        public void divMean(double n){
        this.r/=n;
        this.b/=n;
        this.g/=n;
        }
}
```

## III. QuadTreeImage.java

File ini mengandung code hingga 800 lines sehingga kode sepenuhnya dapat dilihat pada link berikut. Berikut merupakan potongan source code yang relevan dengan algoritma.

```
// Divide and Conquer
public void DnC() {

        this.error = computeError(this);

        // base case
        if (this.size <= getMinBlockSize())
        return;

        if (!errorThresholdCheck(this)) {
        // Useful debugging printer
        // System.out.println("The current block size: " + this.size + " With
start_x: "
        // + this.start_x + " end_x: " + this.end_x + " start_y: " +
this.start_y + "
        // end_y: " + this.end_y);
        int midX = (this.end_x + this.start_x) / 2;
        int midY = (this.end_y + this.start_y) / 2;
        this.ne = new Node(midX, this.end_x, this.start_y, midY);
        this.se = new Node(midX, this.end_x, midY, this.end_y);
        this.sw = new Node(this.start_x, midX, midY, this.end_y);
        this.nw = new Node(this.start_x, midX, this.start_y, midY);
        ne.DnC();
        se.DnC();
        sw.DnC();
```

```
        nw.DnC();
        }
}

public double computeError(Node node) {
        node.meanPixel = meanPixelRange(node.start_x, node.end_x,
node.start_y, node.end_y);
        return switch (this.mode) {
        case 0 -> calcVariance(node.start_x, node.end_x, node.start_y,
node.end_y, node.meanPixel);
        case 1 -> calcMad(node.start_x, node.end_x, node.start_y, node.end_y,
node.meanPixel);
        case 2 -> calcMpd(node.start_x, node.end_x, node.start_y,
node.end_y);
        case 3 -> calcEntropy(node.start_x, node.end_x, node.start_y,
node.end_y);
        case 4 -> calcSsim(node.start_x, node.end_x, node.start_y,
node.end_y, node.meanPixel);
        default -> throw new IllegalArgumentException("Invalid mode: " +
this.mode);
        };
}

public boolean errorThresholdCheck(Node node) {
        return switch (this.mode) {
        case 0 -> node.error <= this.var_thres;
        case 1 -> node.error <= this.mad_thres;
        case 2 -> node.error <= this.mpd_thres;
        case 3 -> node.error <= this.entr_thres;
        case 4 -> node.error <= this.ssim_thres;
        default -> throw new IllegalArgumentException("Invalid mode: " +
this.mode);
        };
}

public void compress() {
        root.DnC();
}

public void applyCompression(boolean logResults) {
        try {
        this.img_output = ImageIO.read(new File(getAbsPath()));
        double start = System.nanoTime();
```

```
      compress();
      double finish = System.nanoTime();
      this.elapsedTime = (finish - start) / 1_000_000_000;
      System.out.println("Compression Elapsed Time: " + this.elapsedTime +
" seconds");
      reconstructImage(logResults);
      this.img_output.flush();
      this.img_output = null;
      } catch (IOException e) {
      System.err.println(e);
      }
}

public void reconstructImage(boolean logResults) {
      reconstructFromNode(root);
      List<Node> treeNodes = getAllNodes();
      int nodeCount = treeNodes.size();
      int treeDepth = getTreeDepth();
      String extension = getExtension();
      String outPath = this.out_path + "." + extension;
      try {
      File outputFile = new File(outPath);
      ImageIO.write(this.img_output, extension, outputFile);
      this.compressedSize = outputFile.length();
      this.compressPercent = 100 - ((this.compressedSize / getFileSize()) *
100); // this is in percentage
      if (logResults) {
            System.out.println("The original size: " + getFileSize() + "
bytes");
            System.out.println("The compressed size: " +
this.compressedSize + " bytes");
            System.out.println("The compression percentage: " +
this.compressPercent + "%");
            System.out.println("Total node count of Quadtree: " +
nodeCount);
            System.out.println("Quadtree depth: " + treeDepth);
      }
      } catch (IOException e) {
      System.out.println(e);
      }
}

public void reconstructFromNode(Node node) {
```

```
      if (node.isLeaf()) {
      // Fill the region with the mean pixel
      for (int y = node.start_y; y <= node.end_y; y++) {
            for (int x = node.start_x; x <= node.end_x; x++) {
                  int r = (int) node.meanPixel.getR();
                  int g = (int) node.meanPixel.getG();
                  int b = (int) node.meanPixel.getB();
                  Color color = new Color(r, g, b);
                  // update output image;
                  this.img_output.setRGB(x, y, color.getRGB());
            }
      }
      } else {
      reconstructFromNode(node.ne);
      reconstructFromNode(node.se);
      reconstructFromNode(node.sw);
      reconstructFromNode(node.nw);
      }
}
```

# BAB IV
# HASIL EKSPERIMEN

## I. Variance Error Method Low Threshold & Min Size Block

**Variance threshold:** 40

**Minimum block size:** 16

**Input Image (2,748,452 bytes) “pexels-eberhardgross-1064162.jpg” :**

**Output Image (2,119,545 bytes) “output_1.jpg” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-eberhardgross-1064162.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output1
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 1
Input variance threshold: 40
Input minimum block size for each node: 16
Start compression [y/n] ?: y
Compression Elapsed Time: 14.151610446 seconds
The original size: 2748452.0 bytes
The compressed size: 2119545.0 bytes
The compression percentage: 22.882226067619143%
Total node count of Quadtree: 2686425
Quadtree depth: 12
Do you want to exit the program [y/n] ?:
```

## II. Variance Error Method High Threshold & Min Size Block

**Variance threshold:** 120

**Minimum block size:** 64

**Input Image (1,205,997 bytes) “pexels-eberhardgross-1064162.jpg” :**

**Output Image  (628,349 bytes) “output_2.jpg” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-eberhardgross-1064162.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_2
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 1
Input variance threshold: 120
Input minimum block size for each node: 64
Start compression [y/n] ?: y
Compression Elapsed Time: 11.527387509 seconds
The original size: 2748452.0 bytes
The compressed size: 1808658.0 bytes
The compression percentage: 34.19357514702823%
Total node count of Quadtree: 575957
Quadtree depth: 11
Do you want to exit the program [y/n] ?:
```

## III. Mean Absolute Deviation Error Method Low Threshold & Min Size Block

**MAD threshold:** 30

**Minimum block size:** 4

**Input Image (2,855,356 bytes) “pexels-lkloeppel-466685.jpg” :**

**Output Image  (1,808,658 bytes) “output_3.jpg” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-lkloeppel-466685.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_3
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 2
Input MAD threshold: 30
Input minimum block size for each node: 4
Start compression [y/n] ?: y
Compression Elapsed Time: 11.171241468 seconds
The original size: 2855356.0 bytes
The compressed size: 1886408.0 bytes
The compression percentage: 33.93440257537064%
Total node count of Quadtree: 1749897
Quadtree depth: 14
Do you want to exit the program [y/n] ?:
```

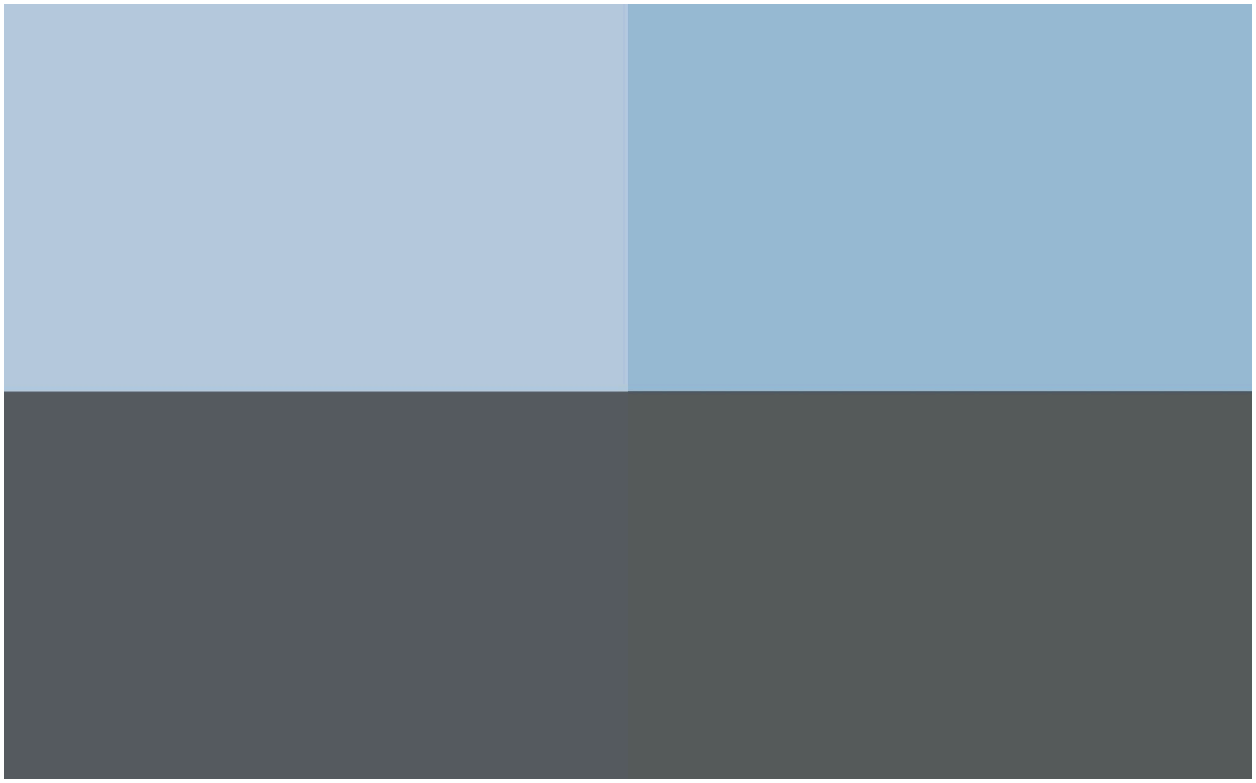
## IV. Mean Absolute Deviation Error Method High Threshold & Min Size Block

**MAD threshold:** 150

**Minimum block size:** 96

**Input Image (2,855,356 bytes) “pexels-lkloeppel-466685.jpg” :**

**Output Image  (345,245 bytes) “output_4.jpg” :**



**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-lkloeppel-466685.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_4
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 2
Input MAD threshold: 150
Input minimum block size for each node: 96
Start compression [y/n] ?: y
Compression Elapsed Time: 2.882848872 seconds
The original size: 2855356.0 bytes
The compressed size: 345245.0 bytes
The compression percentage: 87.90886320304719%
Total node count of Quadtree: 5
Quadtree depth: 2
Do you want to exit the program [y/n] ?:
```

## V. Max Pixel Difference Error Method Low Threshold & Min Size Block

**MPD threshold:** 20

**Minimum block size:** 4

**Input Image (3,700,088 bytes) “pexels-pixabay-35196.png” :**

**Output Image (1,642,670 bytes) “output_5.png” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-pixabay-35196.png
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_5
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 3
Input MPD threshold: 20
Input minimum block size for each node: 4
Start compression [y/n] ?: y
Compression Elapsed Time: 2.397293369 seconds
The original size: 3700088.0 bytes
The compressed size: 1642670.0 bytes
The compression percentage: 55.60456940483577%
Total node count of Quadtree: 600573
Quadtree depth: 13
Do you want to exit the program [y/n] ?:
```

## VI. Max Pixel Difference Error Method High Threshold & Min Size Block

**MPD threshold:** 100

**Minimum block size:** 100

**Input Image (3,700,088 bytes) “pexels-pixabay-35196.png” :**

**Output Image  (252,506 bytes) “output_6.png” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-pixabay-35196.png
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_6
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 3
Input MPD threshold: 100
Input minimum block size for each node: 100
Start compression [y/n] ?: y
Compression Elapsed Time: 1.427211515 seconds
The original size: 3700088.0 bytes
The compressed size: 252506.0 bytes
The compression percentage: 93.17567582176424%
Total node count of Quadtree: 10973
Quadtree depth: 10
Do you want to exit the program [y/n] ?:
```

## VII. Entropy Error Method Low Threshold & Min Size Block

**Entropy threshold:** 2.7

**Minimum block size:** 4

**Input Image (10,839,807 bytes) “pexels-christina99999-31510589.png” :**

**Output Image  (7,822,788 bytes) “output_7.png” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-christina99999-31510589.png
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_7
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 4
Input Entropy threshold: 2.7
Input minimum block size for each node: 16
Start compression [y/n] ?: y
Compression Elapsed Time: 11.945321061 seconds
The original size: 1.0839807E7 bytes
The compressed size: 7822788.0 bytes
The compression percentage: 27.832774144410493%
Total node count of Quadtree: 1733053
Quadtree depth: 13
Do you want to exit the program [y/n] ?:
```

## VIII. Entropy Error Method High Threshold & Min Size Block

**Entropy threshold:** 6.3

**Minimum block size:** 100

**Input Image (10,839,807 bytes) “pexels-christina99999-31510589.png” :**

**Output Image  (595,877 bytes) “output_8.png” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-christina99999-31510589.png
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_8
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 4
Input Entropy threshold: 6.3
Input minimum block size for each node: 100
Start compression [y/n] ?: y
Compression Elapsed Time: 5.134802541 seconds
The original size: 1.0839807E7 bytes
The compressed size: 595877.0 bytes
The compression percentage: 94.50288183175218%
Total node count of Quadtree: 5569
Quadtree depth: 11
Do you want to exit the program [y/n] ?:
```

## IX. SSIM Error Method Low Threshold & Min Size Block

**SSIM threshold:** 0.2

**Minimum block size:** 16

**Input Image (6,402,683 bytes) “pexels-dreamypixel-547119.jpg” :**

**Output Image  (5,691,653 bytes) “output_9.jpg” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-dreamypixel-547119.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_9
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 5
Input SSIM threshold: .2
Input minimum block size for each node: 16
Start compression [y/n] ?: y
Compression Elapsed Time: 24.82339605 seconds
The original size: 6402683.0 bytes
The compressed size: 5691653.0 bytes
The compression percentage: 11.105188246864643%
Total node count of Quadtree: 5503489
Quadtree depth: 12
Do you want to exit the program [y/n] ?:
```

## X. SSIM Error Method High Threshold & Min Size Block

**SSIM threshold:** 0.9

**Minimum block size:** 64

**Input Image (6,402,683 bytes) “pexels-dreamypixel-547119.jpg” :**

**Output Image (4,376,100 bytes) “output_10.jpg” :**

**Terminal CLI:**

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-dreamypixel-547119.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/test/output/output_10
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 5
Input SSIM threshold: .8
Input minimum block size for each node: 64
Start compression [y/n] ?: y
Compression Elapsed Time: 20.685948629 seconds
The original size: 6402683.0 bytes
The compressed size: 4376100.0 bytes
The compression percentage: 31.652090225300867%
Total node count of Quadtree: 1185609
Quadtree depth: 11
Do you want to exit the program [y/n] ?:
```

Untuk data file terlampir pada hasil eksperimen, dapat dilihat pada Github di link berikut.

# BAB V
# ANALISIS PERCOBAAN

Berdasarkan hasil percobaan, didapatkan bahwa nilai error threshold yang mendekati nilai minimum error threshold dengan block size sangat kecil (mendekati empat) menghasilkan gambar yang lebih berkualitas dengan persentase kompresi file size yang berkisar antara nilai 10% - 30%. Akan tetapi, apabila nilai error threshold mendekati nilai maksimum error threshold dengan block size yang relatif besar (seperti 100) menghasilkan gambar yang kurang berkualitas tetapi kompresi file size mencapai angka yang sangat besar hingga 90%. Terkadang terdapat kasus dimana kompresi file sangat besar tetapi bentuk dari gambar tidak ada atau dapat disebut kualitas gambar tidak bagus.

Kompleksitas QuadTree dapat didefinisikan dengan rumus sebagai berikut:

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

$4T(n/2)$ merepresentasikan program melakukan divide and conquer menjadi empat subproblem yang berbeda, sedangkan $O(n^2)$ merepresentasikan operasi kalkulasi rata - rata pixel, error, ataupun operasi metric lainnya pada setiap subproblem. Melalui rumus kompleksitas ini, kita dapat memanfaatkan Master Theorem dimana didapatkan bahwa perumusan dalam bentuk ini:

$$T(n) = a \,.\, T(n/b) + O(n^k)$$

sesuai dengan properti kasus ke-2 Master Theorem dimana $a \log b = k$ sehingga time complexity pada program menjadi:

$$T(n) = \Theta(n^2 \log n)$$

Hasil ini tentunya lebih cepat dibandingkan mengiterasi setiap pixel pada gambar dan melakukan operasi metrics pada setiap individu pixel, sehingga time complexity dapat mencapai $O(n^3)$ .

# BAB VI
# IMPLEMENTASI BONUS

Implementasi bonus pada tugas kecil 2 stima kali ini terdiri atas pencarian persentase kompresi image file dan implementasi error SSIM sebagai metode kalkulasi error.

## I. Find Compression Percentage of given Image File

Mencari persentase kompresi memerlukan pengetahuan atas nilai threshold maksimum dan minimum setiap metode error, dimana hasil perhitungan nilai error. Metode Variance, MAD, dan MPD memiliki nilai threshold minimum 0 dan nilai threshold maksimum 255. Metode Entropy memiliki nilai threshold minimum 2 dan maksimum 8. Metode SSIM memiliki nilai threshold minimum -1, yang dianggap menjadi 0, dan nilai maksimum 1.

Pencarian nilai persentase kompresi menggunakan binary search diantara nilai threshold maksimum dan minimum metode error sehingga ditemukan nilai persentase yang mendekati. Metode yang digunakan untuk bonus membatasi binary search dengan user-input toleransi kesalahan dan maximum iterasi untuk mencegah infinite loop apabila tidak dapat dicapai.

Berikut penjelasan algoritma mencari compression percentage:

1. Menentukan nilai threshold yaitu nilai median diantara min and max threshold dari metode error yang terpilih
2. Lakukan compression
3. Cek apakah file size sudah mencapai compression percentage ± tolerance atau jumlah iterasi melebihi maksimum iterasi. Apabila kondisi terpenuhi maka hentikan pencarian.
4. Apabila kondisi pada langkah 3 tidak terpenuhi, cek apakah persentase compressed lebih tinggi atau lebih rendah dari expected, jika persentase lebih tinggi maka nilai threshold tersebut dibagi dua. Akan tetapi, apabila persentase lebih rendah maka nilai threshold tersebut dikali 1,5.
5. Ulangi pada langkah dua.

Contoh CLI Terminal cari persentase kompresi 60%, toleransi 1, dan max_iter 10 dengan metode variance:

```
nathan@Smallwar20:~/Documents/Github/Tucil2_13523032$ java -cp bin Main
Enter the absolute path of the image: /home/nathan/Documents/Github/Tucil2_13523032/test/input/pexels-dreamypixel-547119.jpg
Image loaded succesfully.
Input absolute output path (without extension): /home/nathan/Documents/Github/Tucil2_13523032/output
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. SSIM
Pick error calculation method: 1
Input variance threshold: 100
Input minimum block size for each node: 16
Start compression [y/n] ?: n
Redirecting to target compression percentage finder...
Input target compression percentage (optional): 60
Input tolerance for target: 1
Input maximum iteration (infinite loop prevention): 10
Compression Elapsed Time: 22.669901741 seconds
Compression Elapsed Time: 22.626942459 seconds
Compression Elapsed Time: 21.542619179 seconds
Compression Elapsed Time: 20.559193956 seconds
Compression Elapsed Time: 19.020888036 seconds
Compression Elapsed Time: 17.886652117 seconds
Compression Elapsed Time: 15.85504367 seconds
Compression Elapsed Time: 12.758729239 seconds
Compression Elapsed Time: 17.152387757 seconds
Compression Elapsed Time: 14.986663277 seconds
Compression Elapsed Time: 11.992391556 seconds
The original size: 6402683.0 bytes
The compressed size: 1873312.0 bytes
The compression percentage: 70.74176560045218%
Total node count of Quadtree: 410813
Quadtree depth: 12
Do you want to exit the program [y/n] ?: n
```

## II. SSIM Error Calculation Method

Berikut rumus yang digunakan untuk menghitung error SSIM dalam image compression:

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

Dengan anggapan kovarians bernilai 0 karena membandingkan bagian daerah yang sama sehingga dapat diabaikan nilai kovariansi pada rumus (program masih melakukan implementasi kovariansi). Lalu nilai error didapatkan dengan 1 dikurang dengan nilai SSIM yang didapatkan berikut:

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

```
return 1 - ((0.299 * ssimR) + (0.587*ssimG) + (0.114*ssimB));
```

# BAB VII
# LAMPIRAN

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

**LINK GITHUB:** https://github.com/19623248Git/Tucil2_13523032