

Laporan Tugas Kecil 2

Ferdinand Gabe Tua Sinaga - 13523051

Untuk kuliah IF2211 Strategi Algoritma

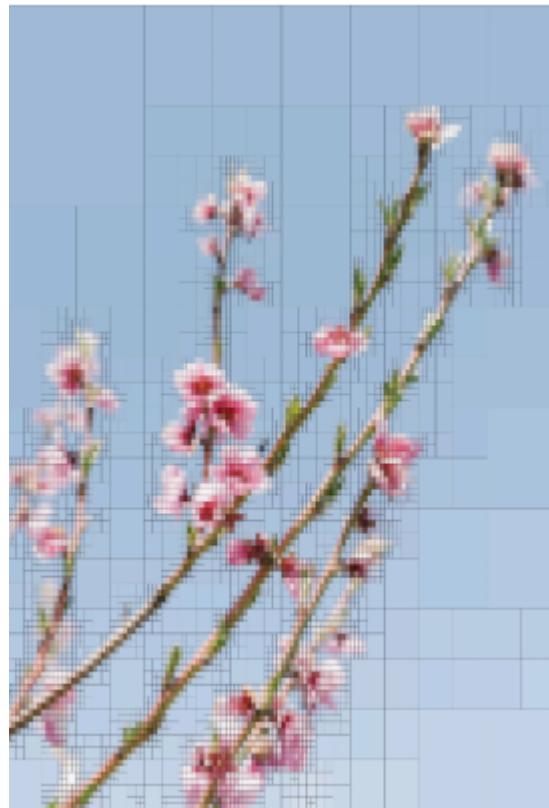
11 April 2025

1. Dasar Teori

Divide and Conquer merupakan salah satu paradigma algoritma yang menyelesaikan suatu permasalahan dengan cara memecah masalah utama menjadi beberapa sub-masalah yang lebih kecil, menyelesaikan masing-masing sub-masalah, lalu menggabungkan hasilnya untuk membentuk solusi akhir dari permasalahan awal. Pendekatan ini dapat diklasifikasikan berdasarkan bagaimana ia membagi suatu permasalahan. Beberapa diantaranya adalah divide by constant (pembagian masalah ke dalam jumlah sub-masalah yang tetap), divide by factor (pembagian masalah berdasarkan faktor ukuran input) , dan divide by Variable (masalah dibagi dengan suatu variabel yang bisa selalu berubah nilainya).

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil. Struktur data tersebut dapat tersebut dapat dimanfaatkan untuk melakukan kompresi terhadap gambar dengan algoritma membagi gambar menjadi 4 area yang lebih kecil secara rekursif tersebut hingga suatu batas eror atau ukuran minimal tercapai.

Berdasarkan karakteristik pembagiannya yang selalu membagi suatu wilayah menjadi 4 bagian tetap, penggunaan Quadtree dalam kompresi gambar merupakan penerapan algoritma divide and conquer dengan pendekatan divide by constant. Hal ini karena jumlah sub-masalah (empat kuadran) selalu tetap pada setiap langkah pembagian, meskipun ukuran sub-masalahnya menjadi lebih kecil seiring proses rekursif berlangsung.



Gambar Kompresi Gambar Oleh Quadtree

2. Algoritma Divide and Conquer

Proses kompresi gambar dengan Quadtree melewati 3 tahap utama yaitu Divide, Conquer, dan Combine. Penjelasan mengenai 3 tahap utama tersebut dapat dilihat dibawah ini.

1. Divide

1. Node akan meninjau satu area persegi pada gambar.
2. Untuk menentukan apakah area tersebut cukup homogen atau tidak, dilakukan perhitungan berdasarkan metrik error yang dipilih oleh pengguna. Beberapa jenis metric yang dapat dipilih user adalah varians, MAD, pixel difference, dan entropy.
3. Dimana perhitungan akan dilakukan dengan melakukan iterasi terhadap seluruh pixel pada area gambar yang dimiliki oleh node itu sendiri dan dibandingkan dengan batasan dari user.
4. Jika ia lebih besar dari batasan eror atau minimum block, ia akan melakukan proses divide lagi menjadi 4 area baru dimana ukuran masing masing area akan memiliki tinggi dan lebar setengah dari parentnya dan mengulangi langkah 1.
5. Jika tidak ia akan lanjut ke proses Conquer

2. Conquer

1. Ketika suatu area sudah lebih kecil dari suatu batasan eror atau minimum block yang di-input oleh user. Ia akan berhenti membagi dirinya sendiri, lalu melakukan normalisasi warna dengan cara mengiterasi seluruh warna pada area tertentu dan mencari rataan dari warnanya.
2. Mencari rataan tersebut dilakukan dengan mengiterasi ke 3 buah warna yaitu merah, hijau, dan biru. Setelah mendapat ke 3 rataan ia akan menyimpan informasi warna tersebut khusus untuk area tersebut saja.
3. Setelah proses ini selesai, algoritma kembali ke pemanggil sebelumnya untuk memproses bagian gambar lainnya yang belum ditinjau.

3. Combine

1. Setelah semua area sudah dieksekusi, seluruh bagian daun yang ada di masing-masing layer pohon (merujuk pada node yang dihasilkan pada setiap kedalaman) akan dicek satu persatu.
2. Dalam pengecekan tersebut setiap daun akan di cek apakah apakah ia memiliki anak atau tidak, hal tersebut dapat diketahui dari atribut yang dibawa oleh setiap daun.
3. Jika ia mempunyai anak, maka iterasi dilanjutkan ke daun selanjutnya dalam layer yang sama, jika ternyata tidak ada yang tidak memiliki anak maka proses akan lanjut ke layer selanjutnya dalam pohon.
4. Jika daun tidak memiliki anak, ia akan diminta untuk melakukan pewarnaan terhadap canvas kosong.
5. Proses ini terus berulang hingga seluruh layer dari pohon ditinjau dan hasilnya akan menjadi 1 buah BufferedImage yang dapat disimpan sebagai gambar.

3. Implementasi

Pada Kompresi gambar menggunakan Quadtree ini saya menggunakan beberapa library java untuk memudahkan proses pengimplementasianya. Berikut library-library tersebut:

1. `java.awt.image.BufferedImage`
2. `java.io.File`
3. `java.io.IOException`
4. `javax.imageio.ImageIO`
5. `java.util.LinkedHashMap`
6. `java.util.List`
7. `java.util.Map`
8. `java.util.ArrayList`
9. `java.util.Scanner`
10. `java.awt.Graphics2D`
11. `java.awt.Color`
12. `com.madgag.gif.fmsware.AnimatedGifEncoder` (Package eksternal)

File utama yang digunakan dalam implementasi ada 3 yaitu:

1. Quadtree.java (tempat implementasi Quadtree)
2. Image.java (tempat implementasi save and load dan compressed image)
3. Main.java (Menerima input user dan menginstansiasi Image untuk kompresi gambar)

Quadtree.java

```
import java.awt.image.BufferedImage;
import java.util.ArrayList;
import java.util.List;
import java.util.Map;
import java.awt.Graphics2D;
import java.awt.Color;

public class Quadtree {
    private int compressedColor;
    private int MetricNum;
    private int x;
    private int y;
    private int width;
    private int height;
    private BufferedImage origin;

    public static int jumlahObjek;
    public static int MIN_SIZE ;
    public static double EROR_TRESHOLD ;

    private Quadtree topLeft = null;
    private Quadtree topRight = null;
    private Quadtree bottomLeft = null;
    private Quadtree bottomRight = null;

    public boolean divided = false;
```

```

public Quadtree(int w, int h, int posX, int posY, BufferedImage imag, int metric){
    this.width = w;
    this.height = h;
    this.x = posX;
    this.y = posY;
    this.origin = imag;
    this.MetricNum = metric;
    jumlahObjek+=1;
}

public boolean canDivide(){
    if(MetricNum == 1){           //variance
        return (getVariance()>EROR_TRESHOLD) && (width*height > MIN_SIZE);
    }
    else if(MetricNum == 2){      //MeanAbsoluteDeviation
        return( getMeanAbsoluteDeviation()>EROR_TRESHOLD )&& (width*height > MIN_SIZE);
    }
    else if(MetricNum == 3){      //MaxPixelDiff
        return (getMaxPixelDifference()>EROR_TRESHOLD) && (width*height > MIN_SIZE);
    }
    else if(MetricNum == 4){      //Entropy
        return (getEntropy()>EROR_TRESHOLD) && (width*height > MIN_SIZE);
    }

    return false;
}

public void subDivide(int steps, Map<Integer, List<Quadtree>> stepsMap){
    normalizeQuadTree();
    stepsMap.computeIfAbsent(steps, k -> new ArrayList<>()).add(this);

    if(canDivide()){

        divided = true;
        int halfWidth = width/ 2;
        int halfHeight = height / 2;
        int rightWidth = width - halfWidth;
        int bottomHeight = height - halfHeight;

        topLeft = new Quadtree(halfWidth, halfHeight, x, y, origin, MetricNum);
        topRight = new Quadtree(rightWidth, halfHeight, x + halfWidth, y, origin, MetricNum);
        bottomLeft = new Quadtree(halfWidth, bottomHeight, x, y + halfHeight, origin, MetricNum);
        bottomRight = new Quadtree(rightWidth, bottomHeight, x + halfWidth, y + halfHeight, origin, MetricNum);

        topLeft.subDivide(steps + 1, stepsMap);
        topRight.subDivide(steps + 1, stepsMap);
        bottomLeft.subDivide(steps + 1, stepsMap);
        bottomRight.subDivide(steps + 1, stepsMap);
    }
    else{
    }
}

```

```
public int getWidth(){return this.width;} ;
public int getHeight(){return this.height;} ;

//Error Metric
public double getVariance(){
    int[] sum = {0, 0, 0};
    double[] avg = {0.0, 0.0, 0.0};
    double[] var = {0.0, 0.0, 0.0};
    double totalPixels = width * height;

    for (int i = x; i < x + width; i ++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            sum[0] = sum[0] + ((rgb >> 16) & 0xFF); // Merah
            sum[1] = sum[1] + ((rgb >> 8) & 0xFF); // Hijau
            sum[2] = sum[2] + (rgb & 0xFF); //Biru
        }
    }
    avg[0] = sum[0] / totalPixels;
    avg[1] = sum[1] / totalPixels;
    avg[2] = sum[2] / totalPixels;

    for (int i = x; i < x + width; ++i) {
        for (int j = y; j < y + height; ++j) {
            int val = origin.getRGB(i,j);
            int red = (val >> 16) & 0xFF;
            int green = (val >> 8) & 0xFF;
            int blue = val & 0xFF;

            var[0] += (red - avg[0]) * (red - avg[0]);
            var[1] += (green - avg[1]) * (green - avg[1]);
            var[2] += (blue - avg[2]) * (blue - avg[2]);
        }
    }
    var[0] /= totalPixels;
    var[1] /= totalPixels;
    var[2] /= totalPixels;

    double avgvar = (var[0] + var[1] + var[2]) / 3.0;
}

return avgvar;
}
```

```
public double getMeanAbsoluteDeviation(){
    int[] sum = {0, 0, 0};
    double[] avg = {0.0, 0.0, 0.0};
    double[] mad = {0.0, 0.0, 0.0};
    double totalPixels = width * height;

    for (int i = x; i < x + width; i++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            sum[0] = sum[0] + ((rgb >> 16) & 0xFF); //merah
            sum[1] = sum[1] + ((rgb >> 8) & 0xFF); //hijau
            sum[2] = sum[2] + (rgb & 0xFF);           //biru
        }
    }
    avg[0] = sum[0] / totalPixels;
    avg[1] = sum[1] / totalPixels;
    avg[2] = sum[2] / totalPixels;

    for (int i = x; i < x + width; i++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            int merah = (rgb >> 16) & 0xFF; //merah
            int green = (rgb >> 8) & 0xFF;  //hijau
            int blue  = rgb & 0xFF;         //biru
        }
    }
}
```

```
    avg[0] = sum[0] / totalPixels;
    avg[1] = sum[1] / totalPixels;
    avg[2] = sum[2] / totalPixels;

    for (int i = x; i < x + width; i ++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            int merah = (rgb >> 16) & 0xFF; //merah
            int green = (rgb >> 8) & 0xFF; //hijau
            int blue = rgb & 0xFF; //biru

            mad[0] += Math.abs(merah - avg[0]);
            mad[1] += Math.abs(green - avg[1]);
            mad[2] += Math.abs(blue - avg[2]);
        }
    }

    mad[0] /= totalPixels;
    mad[1] /= totalPixels;
    mad[2] /= totalPixels;

    return (mad[0] + mad[1] + mad[2]) / 3.0;
}
```

```
public double getMaxPixelDifference(){
    int maxColor[] = {0,0,0};
    int minColor[] = {255,255,255};

    for (int i = x; i < x + width; i ++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            int merah = (rgb >> 16) & 0xFF; //merah
            int green = (rgb >> 8) & 0xFF; //hijau
            int blue = rgb & 0xFF; //biru

            if (maxColor[0]<merah){
                maxColor[0]=merah;
            }
            if (maxColor[1]<green){
                maxColor[1]=green;
            }
            if (maxColor[2]<blue){
                maxColor[2]=blue;
            }
            if (minColor[0]>merah){
                minColor[0]=merah;
            }
            if (minColor[1]>green){
                minColor[1]=green;
            }

            if (minColor[2]>blue){
                minColor[2]=blue;
            }
        }
    }

    return
        ((maxColor[0]-minColor[0]) +
        (maxColor[1]-minColor[1]) +
        (maxColor[2]-minColor[2])) / 3;
}
```

```
public double getEntropy(){
    int[] redHist = new int[256];
    int[] greenHist = new int[256];
    int[] blueHist = new int[256];
    double[] entropy = {0.0, 0.0, 0.0};

    for (int j = y; j < y+ height; j++) {
        for (int i = x; i < x+width; i++) {
            int pixel = origin.getRGB(i, j);

            int red = (pixel >> 16) & 0xFF;
            int green = (pixel >> 8) & 0xFF;
            int blue = pixel & 0xFF;

            redHist[red]++;
            greenHist[green]++;
            blueHist[blue]++;
        }
    }

    for (int i = 0; i < 256; i++) {
        if (redHist[i] > 0) {
            double p = (double) redHist[i] / (width*height);
            entropy[0] += -p * (Math.log(p) / Math.log(2));
        }
        if (greenHist[i] > 0) {
            double p = (double) greenHist[i] / (width*height);
            entropy[1] += -p * (Math.log(p) / Math.log(2));
        }
        if (blueHist[i] > 0) {
            double p = (double) blueHist[i] / (width*height);
            entropy[2] += -p * (Math.log(p) / Math.log(2));
        }
    }
    double a = (entropy[0] + entropy[1] + entropy[2]) / 3.0;
    return a;
}

public void setOrigin(Bitmap origin) {
```

```
public void normalizeQuadTree(){
    int[] sum = {0, 0, 0};
    double[] avg = {0.0, 0.0, 0.0};
    double totalPixels = width * height;

    for (int i = x; i < x + width; i++){
        for (int j = y; j < y + height; j++){
            int rgb = origin.getRGB(i, j);
            sum[0] = sum[0] + ((rgb >> 16) & 0xFF); //merah
            sum[1] = sum[1] + ((rgb >> 8) & 0xFF); //hijau
            sum[2] = sum[2] + (rgb & 0xFF); //biru
        }
    }
    avg[0] = sum[0] / totalPixels;
    avg[1] = sum[1] / totalPixels;
    avg[2] = sum[2] / totalPixels;

    int avgRed = (int) avg[0];
    int avgGreen = (int) avg[1];
    int avgBlue = (int) avg[2];
    int alpha = 255;

    compressedColor = (alpha << 24) | (avgRed << 16) | (avgGreen << 8) | avgBlue;
}
```

```
public void drawToCanvas(BufferedImage canvas) {
    Graphics2D g = canvas.createGraphics();
    g.setColor(new Color(
        (compressedColor >> 16) & 0xFF,
        (compressedColor >> 8) & 0xFF,
        compressedColor & 0xFF
    ));
    g.fillRect(x, y, width, height);
    g.dispose();
}
```

Image.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;
import java.util.LinkedHashMap;
import java.util.List;
import java.util.Map;
import java.awt.Graphics2D;
import com.madgag.gif.fmsware.AnimatedGifEncoder;

public class Image{
    public double sizeBefore;
    public double sizeAfter;

    private BufferedImage original;
    private BufferedImage canvas;
    private int width;
    private int height;
    private String inputExtension;
    private boolean failToLoad;

    public Image(String filename) {
        try {

            File inputFile = new File(filename);
            long inputSize = inputFile.length();
            this.sizeBefore = inputSize;

            this.original = ImageIO.read(inputFile);
            this.width = original.getWidth();
            this.height = original.getHeight();
            this.canvas = new BufferedImage(width, height, original.getType());

            int dotIndex = filename.lastIndexOf('.');
            if (dotIndex != -1 && dotIndex < filename.length() - 1) {
                this.inputExtension = filename.substring(dotIndex + 1).toLowerCase();
                System.out.println("Gambar berhasil dimuat dengan ekstensi: " + inputExtension);
            } else {
                throw new IllegalArgumentException(
                    "Tolong berikan path absolut + ekstensi dari filenya, contoh: C:/gambar/foto.jpeg"
                );
            }

        } catch (IOException e) {
            this.failToLoad = true;
            System.out.println("Terjadi kesalahan saat memuat gambar: " + e.getMessage());
        } catch (IllegalArgumentException e) {
            this.failToLoad = true;
            System.out.println("Kesalahan input: " + e.getMessage());
        }
    }
}
```

```
public void saveImage(String outputPath) {
    try {
        File output = new File(outputPath);
        File parentDir = output.getParentFile();

        if (parentDir != null && !parentDir.exists()) {
            if (parentDir.mkdirs()) {
                System.out.println("Direktori dibuat: " + parentDir.getAbsolutePath());
            } else {
                System.err.println("Gagal membuat direktori: " + parentDir.getAbsolutePath());
                return;
            }
        }

        String fileName = output.getName();
        int dotIndex = fileName.lastIndexOf('.');
        if (dotIndex == -1 || dotIndex == fileName.length() - 1) {
            System.err.println("Ekstensi file tidak valid. Contoh yang benar: /path/to/output.jpeg");
            return;
        }
    }
}
```

```
String format = fileName.substring(dotIndex + 1).toLowerCase();
BufferedImage imageToSave = canvas;
if (format.equals("jpg") || format.equals("jpeg")) {
    imageToSave = new BufferedImage(canvas.getWidth(), canvas.getHeight(), BufferedImage.TYPE_INT_RGB);
    Graphics2D g = imageToSave.createGraphics();
    g.drawImage(canvas, x:0, y:0, observer:null);
    g.dispose();
}

boolean result = ImageIO.write(imageToSave, format, output);
if (result) {
    System.out.println("Gambar Berhasil Disimpan Pada " + output.getAbsolutePath());
    System.out.println("GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar");
    long outputSize = output.length();
    this.sizeAfter = outputSize;

} else {
    System.err.println("Format " + format + " tidak didukung.");
}

} catch (IOException e) {
    System.err.println("Gagal menyimpan gambar: " + e.getMessage());
}
}
```

```
public void compressImage(String gifOutputPathWithoutExt, int metricNum, double threshold) {
    if (failToLoad) return;

    Quadtree root = new Quadtree(width, height, posX:0, posY:0, original, metricNum);
    Map<Integer, List<Quadtree>> stepsMap = new LinkedHashMap<>();
    root.subDivide(steps:0, stepsMap);

    int panjang = stepsMap.size();
    for (int i = 0; i < panjang - 1; i++) {
        List<Quadtree> quadList = stepsMap.get(i);
        for (Quadtree value : quadList) {
            if (!value.divided) {
                stepsMap.get(i + 1).add(value);
            }
        }
    }

    try {
        AnimatedGifEncoder gifEncoder = new AnimatedGifEncoder();
        gifEncoder.start(gifOutputPathWithoutExt + ".gif");
        gifEncoder.setDelay(ms:500);
        gifEncoder.setRepeat(iter:0);

        for (Map.Entry<Integer, List<Quadtree>> step : stepsMap.entrySet()) {
            Graphics2D g = canvas.createGraphics();
            g.drawImage(original, x:0, y:0, observer:null);
            g.dispose();

            for (Quadtree node : step.getValue()) {
                node.drawToCanvas(canvas);
            }

            gifEncoder.addFrame(canvas);
        }

        gifEncoder.finish();
        saveImage(gifOutputPathWithoutExt + "." + inputExtension);
        System.out.println(x:"");
        System.out.println("Size Gambar sebelum dikompreksi adalah "+sizeBefore);
        System.out.println("Size Gambar setelah dikompreksi adalah "+ sizeAfter);
        System.out.println("Percentase Kompreksi adalah "+(1 - ((double) sizeAfter / sizeBefore)) * 100 + "%");
        System.out.println("Kedalaman Pohon adalah " + stepsMap.size());
        System.out.println("Banyak Pohon adalah " + Quadtree.jumlahObjek);
    } catch (Exception e) {
        System.err.println("Gagal membuat GIF: " + e.getMessage());
    }
}
```

Main.java

```
import java.util.Scanner;

public class Main {
    Run | Debug
    public static void main(String[] args) {
        Scanner scanner = new Scanner(System.in);

        System.out.println(x:"== Kompresi Gambar dengan Quadtree ==");
        System.out.print(s:"Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): ");
        String inputPath = scanner.nextLine().trim();

        System.out.print(s:"Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): ");
        String outputPath = scanner.nextLine().trim();

        System.out.println(x:"Pilih metric:");
        System.out.println(x:"1: Variance");
        System.out.println(x:"2: Mean Absolute Deviation (MAD)");
        System.out.println(x:"3: Max Pixel Difference");
        System.out.println(x:"4: Entropy");
        System.out.print(s:"Masukkan nomor metric (1-4): ");
        int metricNum = 0;
        while (true) {
            if (scanner.hasNextInt()) {
                metricNum = scanner.nextInt();
                if (metricNum >= 1 && metricNum <= 4) {
                    break;
                }
            }
            System.out.println(x:"Nomor metric tidak valid. Harus antara 1 - 4.");
            scanner.nextLine();
        }

        double threshold = -1;
        while (true) {
            System.out.print(s:"Masukkan error threshold (misal: 500.0): ");
            if (scanner.hasNextDouble()) {
                threshold = scanner.nextDouble();
                boolean valid = false;

                switch (metricNum) {
                    case 1:
                        valid = threshold >= 0;
                        if (!valid) System.out.println(x:"Threshold untuk Variance harus >= 0.");
                        break;
                    case 2:
                        valid = threshold >= 0 && threshold <= 255;
                        if (!valid) System.out.println(x:"Threshold untuk MAD harus antara 0 - 255.");
                        break;
                    case 3:
                        valid = threshold >= 0 && threshold <= 255;
                        if (!valid) System.out.println(x:"Threshold untuk Max Pixel Difference harus antara 0 - 255.");
                        break;
                    case 4:
                        valid = threshold >= 1 && threshold <= 8;
                        if (!valid) System.out.println(x:"Threshold untuk Entropy harus antara 1 - 8.");
                        break;
                }
            }
        }
    }
}
```

```
        if (valid){
            break;
        }
    } else {
        scanner.next();
    }

    System.out.println("Input tidak valid. Coba lagi.");
}

int minBlockSize = 0;
while (true) {
    System.out.print("Masukkan minimum block size (misal: 4): ");
    if (scanner.hasNextInt()) {
        minBlockSize = scanner.nextInt();
        if (minBlockSize >= 1) {
            break;
        }
    }
    System.out.println("Block size tidak valid. Masukkan angka >= 1.");
    scanner.nextLine();
}
scanner.close();

Image image = new Image(inputPath);
if (!image.isFailToLoad()) {
    Quadtree.ERROR_THRESHOLD=threshold;
    Quadtree.MIN_SIZE = minBlockSize;
    System.err.println("Processing . . .");
    System.out.println();
    long startTime = System.currentTimeMillis();
    image.compressImage(outputPath, metricNum, threshold);
    long endTime = System.currentTimeMillis();

    long duration = endTime - startTime;
    System.out.println("Waktu eksekusi: " + duration + " ms");

} else {
    System.out.println("Gagal memuat gambar. Program dihentikan.");
}
}
```

4. Test

Test 1

```
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input.png
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output1
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 1
Masukkan error threshold (misal: 500.0): 200
Masukkan minimum block size (misal: 4): 64
Gambar berhasil dimuat dengan ekstensi: png
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output1.png
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 1295659.0
Size Gambar setelah dikompresi adalah 108374.0
Percentase Kompresi adalah 91.63560782582454%
Kedalaman Pohon adalah 9
Banyak Pohon adalah 16313
Waktu eksekusi: 2933 ms
```





Test 2

```
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input3.png
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output2
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 2
Masukkan error threshold (misal: 500.0): 10
Masukkan minimum block size (misal: 4): 5
Gambar berhasil dimuat dengan ekstensi: png
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output2.png
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 83780.0
Size Gambar setelah dikompresi adalah 35627.0
Percentase Kompresi adalah 57.47553115301982%
Kedalaman Pohon adalah 9
Banyak Pohon adalah 13701
Waktu eksekusi: 267 ms
```





Test 3

```
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input4.png
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output3
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 4
Masukkan error threshold (misal: 500.0): 1.5
Masukkan minimum block size (misal: 4): 8
Gambar berhasil dimuat dengan ekstensi: png
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output3.png
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 332505.0
Size Gambar setelah dikompresi adalah 183286.0
Percentase Kompresi adalah 44.87721989143021%
Kedalaman Pohon adalah 9
Banyak Pohon adalah 74049
Waktu eksekusi: 844 ms
```





Test 4

```
Kompresi Gambar dengan Python
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input5.jpg
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output4
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 3
Masukkan error threshold (misal: 500.0): 8
Masukkan minimum block size (misal: 4): 32
Gambar berhasil dimuat dengan ekstensi: jpg
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output4.jpg
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 463292.0
Size Gambar setelah dikompresi adalah 30656.0
Persentase Kompresi adalah  93.38300682938622%
Kedalaman Pohon adalah 8
Banyak Pohon adalah 17149
Waktu eksekusi: 671 ms
```





Test 5

```
Kompresi gambar dengan quanlitas 60
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input8.jpg
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output5
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 4
Masukkan error threshold (misal: 500.0): 2
Masukkan minimum block size (misal: 4): 8
Gambar berhasil dimuat dengan ekstensi: jpg
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output5.jpg
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 80386.0
Size Gambar setelah dikompresi adalah 68635.0
Persentase Kompresi adalah 14.618217102480536%
Kedalaman Pohon adalah 10
Banyak Pohon adalah 191521
Waktu eksekusi: 2166 ms
```



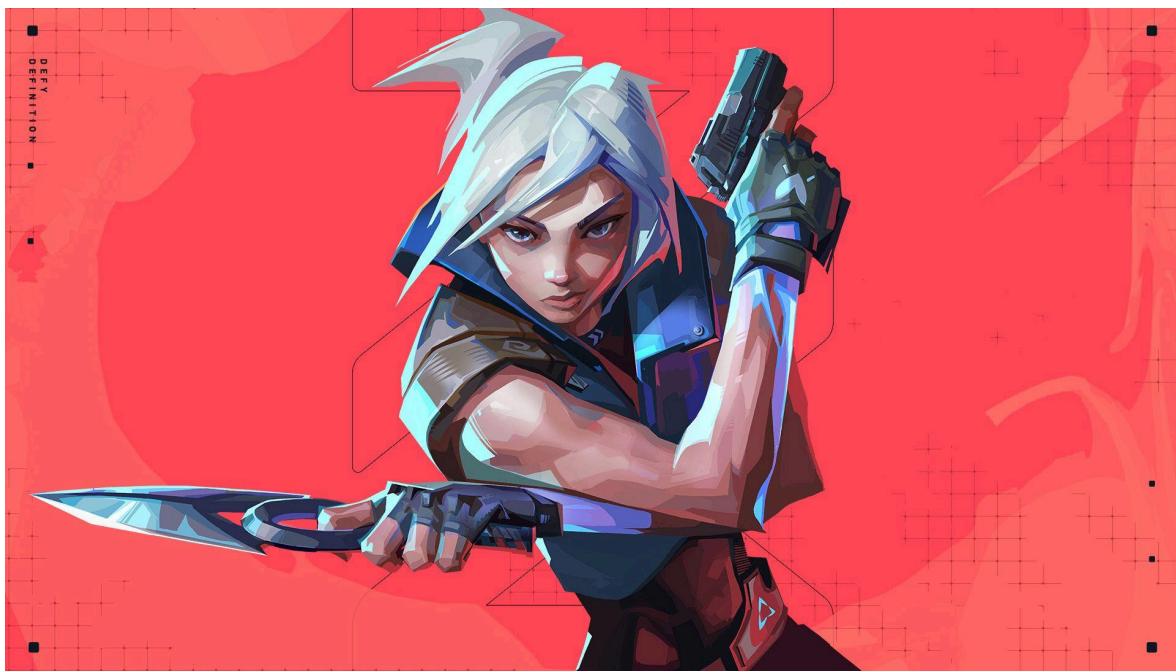


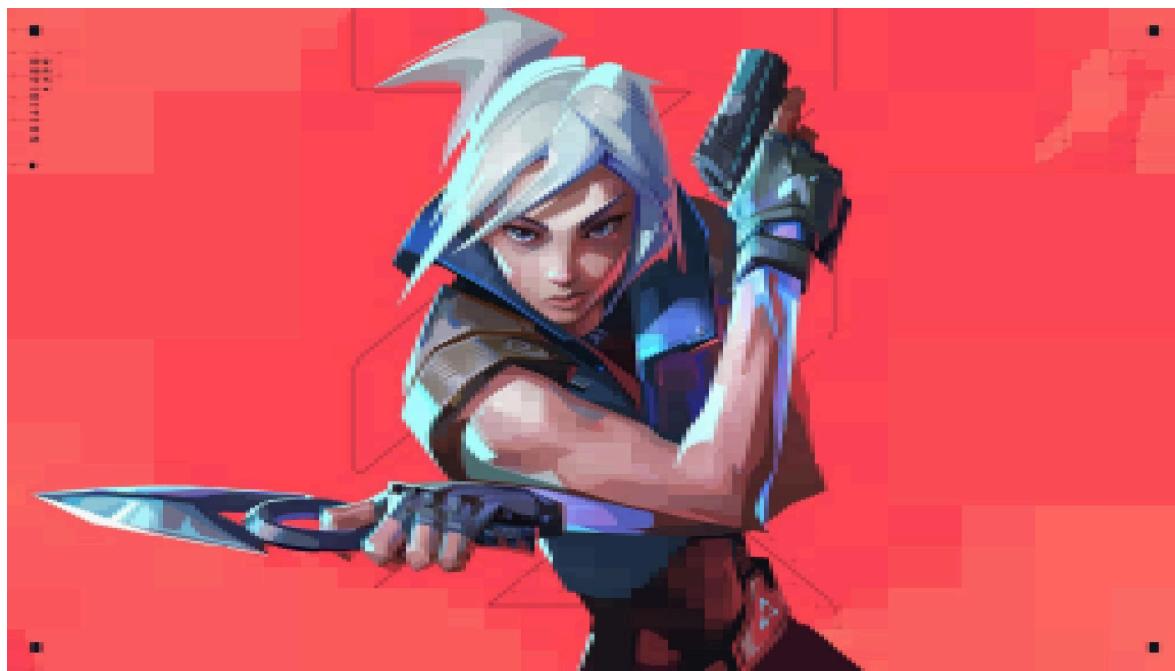
Test 6

```
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input9.jpg
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output6
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 1
Masukkan error threshold (misal: 500.0): 100
Masukkan minimum block size (misal: 4): 64
Gambar berhasil dimuat dengan ekstensi: jpg
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output6.jpg
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 183818.0
Size Gambar setelah dikompresi adalah 128747.0
Persentase Kompresi adalah 29.95952518251749%
Kedalaman Pohon adalah 9
Banyak Pohon adalah 21173
Waktu eksekusi: 2966 ms
```





Test 7

```
Masukkan path absolut ke file gambar (misal: C:/images/foto.jpg): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\input10.jpg
Masukkan path absolut output tanpa ekstensi (misal: C:/images/output): D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output7
Pilih metric:
1: Variance
2: Mean Absolute Deviation (MAD)
3: Max Pixel Difference
4: Entropy
Masukkan nomor metric (1-4): 2
Masukkan error threshold (misal: 500.0): 10
Masukkan minimum block size (misal: 4): 32
Gambar berhasil dimuat dengan ekstensi: jpg
Processing . . .

Gambar Berhasil Disimpan Pada D:\STIMA\TUCIL BUAT 11 MARET\Tucil2_13523051\output7.jpg
GIF Berhasil Disimpan pada Folder yang Sama Dengan Gambar

Size Gambar sebelum dikompresi adalah 98676.0
Size Gambar setelah dikompresi adalah 35353.0
Percentase Kompresi adalah 64.1726458307998%
Kedalaman Pohon adalah 8
Banyak Pohon adalah 9797
Waktu eksekusi: 606 ms
```



5. Analisis

Pada bagian ini, kita akan menganalisis kompleksitas waktu dari algoritma pembentukan Quadtree dalam proses kompresi gambar. Algoritma ini mengikuti paradigma Divide and Conquer, dan kompleksitas waktunya ditentukan oleh beberapa 3 proses utama yaitu:

1. Pengecekan error untuk setiap sub-area

Pada setiap node (area gambar), algoritma akan melakukan perhitungan berdasarkan metrik error tertentu (seperti varians, MAD, pixel difference, atau entropi). Proses ini melibatkan iterasi terhadap seluruh piksel dalam sub-area tersebut sehingga membutuhkan waktu sebesar $3n$ dimana n merupakan banyak piksel pada area tersebut.

2. Proses pembagian menjadi 4 bagian (Divide)

Jika error melebihi ambang batas atau ukuran lebih besar dari minimum block, area akan dibagi menjadi 4 sub-area dengan ukuran lebih kecil. Proses pembagian ini dilakukan secara rekursif, membentuk struktur pohon dengan banyak node. Secara umum, proses rekursif ini akan memanggil 4 subarea sehingga bisa ditulis sebagai berikut $4T(n/4)$.

3. Proses penyatuan kembali hasil (Combine)

Setelah seluruh node diproses, algoritma akan menggabungkan hasil dari setiap bagian dan melakukan rendering pada canvas akhir. Proses ini akan memakan waktu sebesar n untuk setiap luas area yang akan diwarnai

Berdasarkan 3 tahapan diatas dapat diketahui bahwa ekspresi matematika kompleksitas waktunya adalah

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

Persamaan tersebut memenuhi teorema Master sehingga kita bisa menghitung Big Onya menggunakan case 2 dan mendapatkan Big O sebesar $O(n \log n)$.

6. Implementasi Bonus

Pada tugas ini, saya mengimplementasikan fitur bonus berupa pembuatan animasi GIF yang menampilkan proses pembentukan Quadtree secara bertahap. Proses ini dimulai dengan memetakan setiap node Quadtree yang terbentuk ke dalam sebuah struktur Map, di mana setiap entri memiliki key berupa kedalaman dimana node tersebut terbentuk, dan value berupa daftar node yang terbentuk pada tahap tersebut. Setelah itu, saya melakukan iterasi terhadap setiap tahap dan jika ditemukan node yang belum memiliki anak (daun), node tersebut akan dipindahkan ke kedalaman berikutnya proses tersebut akan terus berlangsung hingga key ke n-1. Selanjutnya saya membuat image untuk masing masing key dalam map tersebut dan menggabungkannya untuk membuat GIF. Berikut snapshot implementasinya:

```
AnimatedGifEncoder gifEncoder = new AnimatedGifEncoder();
gifEncoder.start(gifOutputPathWithoutExt + ".gif");
gifEncoder.setDelay(ms:500);
gifEncoder.setRepeat(iter:0);

for (Map.Entry<Integer, List<Quadtree>> step : stepsMap.entrySet()) {
    Graphics2D g = canvas.createGraphics();
    g.drawImage(original, x:0, y:0, observer:null);
    g.dispose();

    for (Quadtree node : step.getValue()) {
        node.drawToCanvas(canvas);
    }

    gifEncoder.addFrame(canvas);
}

gifEncoder.finish();
```

Gambar implementasi bonus GIF

7. Lampiran

Pranala Github : https://github.com/Ferdinand18780/Tucil2_13523051