Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Pemanfaatan Algoritma *Divide and Conquer* Dalam Pengkompresian Gambar

Disusun oleh:

Muhammad Zakkiy (10122074)
Dzubyan Ilman Ramadhan (10122010)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025

Algoritma Divide and Conquer

Input : threshold, ukuran blok minimal, gambar input

Output : gambar output

1. Input threshold, ukuran blok minimal, dan gambar
2. Inisiasi Quadtree dengan variabel koordinat x dan y dari tiap upa-blok, ukuran blok (array 2 nilai), warna rata-rata, dan boolean isLeaf untuk menentukan apakah suatu simpul adalah daun (tidak memiliki simpul anak)
3. Hitung pengukuran error dengan metode yang telah dipilih sebelumnya dan area dari perkalian ukuran blok.
4. a. Apabila hasil pengukuran error melebihi threshold dan ukuran blok setelah dibagi menjadi 4 tidak akan menjadi kurang ukuran blok minimal, blok tersebut akan dibagi menjadi 4. Tambahkan 4 anak simpul pada simpul representasi blok tersebut pada quadtree.

   b. Pada tiap anak simpul, simpan ukuran upa-blok, yaitu ukuran simpul orang tuanya dibagi 2. Untuk mencegah bilangan desimal, bulatkan salah satu upa-blok ke atas apabila ukuran blok orang tuanya adalah ganjil. Simpan juga koordinat dari pojok kiri tiap upa-blok pada simpulnya.

   c. Lakukan langkah 3, 4 dan 5 untuk tiap simpul tersebut.
5. Apabila tidak memenuhi syarat pada 4a, sebuah blok dengan warna rata-rata dari warna semua piksel di blok tersebut dan ukuran yang tercatat pada representasi simpul akan di render ke output gambar.
6. Simpan output gambar yang telah dibuat.

Source Code

```
import javax.imageio.ImageIO; //IO Gambar

import java.awt.image.BufferedImage; //Manipulasi Gambar & Tipe Data BufferedImage

import java.io.File; //Manipulasi File

import java.io.IOException; //Representasi File

import java.awt.Color; //Tipe data warna

import java.awt.Graphics2D; //Rendering Gambar

import java.util.Scanner; //Input CLI

import java.util.Locale;




public class Quadtree {

    //Variabel Input
```

```
    static String absolute_address_in = "";
    static String error_method = "";
    static double threshold = 0;
    static int min_size = 0;
    static String absolute_address_out = "";
    static BufferedImage img = new BufferedImage(1, 1, BufferedImage.TYPE_INT_RGB);
    static int target = 0;
    //static String absolute_address_gif_out = "";


    static class QuadtreeNode {
        int x = 0, y = 0;
        int[] size = {0, 0};
        Color avgColor = new Color(0, 0, 0);
        boolean isLeaf;
        QuadtreeNode[] children = new QuadtreeNode[4];


        QuadtreeNode(int x, int y, int[] size, Color avgColor, boolean isLeaf) {
            this.x = x;
            this.y = y;
            this.size = size;
            this.avgColor = avgColor;
            this.isLeaf = isLeaf;
        }


        public int getDepth() {
            if (isLeaf) {
                return 1;
            }


            int max = 0;
            for(QuadtreeNode child : children) {
                if (child != null) {
                    max = Math.max(max, child.getDepth());
                }
```

```
            }

            return 1 + max;
        }

        public long getNodeTotal() {
            if (isLeaf) {
                return 1;
            }

            int nodeCount = 0;
            for(QuadtreeNode child : children) {
                if (child != null) {
                    nodeCount += child.getNodeTotal();
                }
            }

            return nodeCount;
        }

        public void drawImageByDepth(Graphics2D g, int targetDepth) {
            int[] size = {img.getHeight(), img.getWidth()};
            drawByDepthHelper(g, 0, 0, size, 0, targetDepth);
        }

        private void drawByDepthHelper(Graphics2D g, int x, int y, int[] size, int
currentDepth, int targetDepth) {
            if (currentDepth == targetDepth || (children[0] == null)) {
                Color avg = get_average_color(x, y, size);
                g.setColor(avg);
                g.fillRect(y, x, size[1], size[0]);
                return;
            }

            if (children[0] != null) {
```

```
                int[] new_size_1 = {size[0] / 2, size[1] / 2};
                int[] new_size_2 = {size[0] / 2, size[1] / 2};
                int[] new_size_3 = {size[0] / 2, size[1] / 2};
                int[] new_size_4 = {size[0] / 2, size[1] / 2};

                if (size[0] % 2 != 0 && size[1] % 2 == 0) {
                    new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                    new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2;
                    new_size_3[0] = size[0] / 2 + 1; new_size_3[1] = size[1] / 2;
                    new_size_4[0] = size[0] / 2 + 1; new_size_4[1] = size[1] / 2;
                } else if (size[0] % 2 == 0 && size[1] % 2 != 0) {
                    new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                    new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2 + 1;
                    new_size_3[0] = size[0] / 2; new_size_3[1] = size[1] / 2;
                    new_size_4[0] = size[0] / 2; new_size_4[1] = size[1] / 2 + 1;
                } else if (size[0] % 2 != 0 && size[1] % 2 != 0) {
                    new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                    new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2 + 1;
                    new_size_3[0] = size[0] / 2 + 1; new_size_3[1] = size[1] / 2;
                    new_size_4[0] = size[0] / 2 + 1; new_size_4[1] = size[1] / 2 + 1;
                }

                children[0].drawByDepthHelper(g, x, y, new_size_1, currentDepth + 1,
targetDepth);
                children[1].drawByDepthHelper(g, x, y + new_size_1[1], new_size_2,
currentDepth + 1, targetDepth);
                children[2].drawByDepthHelper(g, x + new_size_1[0], y, new_size_3,
currentDepth + 1, targetDepth);
                children[3].drawByDepthHelper(g, x + new_size_1[0], y + new_size_1[1],
new_size_4, currentDepth + 1, targetDepth);
            }
        }
    }

    static int[] _size = {0, 0};
    static Color _color = new Color(0, 0, 0);
```

```
    static QuadtreeNode tree = new QuadtreeNode(0, 0, _size, _color, false);


    public static void main(String[] args) {

        long mulai = System.currentTimeMillis();


        init();

        int[] size = {img.getHeight(), img.getWidth()};

        tree = block_division(0, 0, size);

        BufferedImage img_out = make_images(tree.getDepth() - 1);

        save_and_out(img_out);


        long selesai = System.currentTimeMillis();

        long durasi = selesai - mulai;


        File img_file_in = new File(absolute_address_in);

        File img_file_out = new File(absolute_address_out);


        long sizeOriginal = img_file_in.length();

        long sizeCompressed = img_file_out.length();


        System.out.println("Ukuran gambar input : " + writeSize(sizeOriginal));

        System.out.println("Ukuran gambar output : " + writeSize(sizeCompressed));

        float compressPercentage = (1 - (float) sizeCompressed / sizeOriginal)*100;

        System.out.println("Persentase kompresi : " + compressPercentage + " %");

        System.out.println("Waktu eksekusi: " + durasi + " ms");

        System.out.println("Kedalaman simpul dalam implementasi : " + tree.getDepth() + "
simpul");

        System.out.println("Banyak simpul dalam implementasi : " + tree.getNodeTotal() + "
simpul");

    }


    public static void init() { //Initialisasi input

        Scanner scanner = new Scanner(System.in);

        scanner.useLocale(Locale.US);
```

```
        System.out.print("Alamat absolut input: ");
        absolute_address_in = scanner.nextLine().trim();

        System.out.print("Metode perhitungan \nPilihan : ");
        System.out.println("Variance (var) | Mean Absolute Deviation (mad) | Max Pixel
Difference (mpd) | Entropy (ent)");
        System.out.print(": ");
        error_method = scanner.nextLine().trim();

        System.out.print("Alamat absolut output: ");
        absolute_address_out = scanner.nextLine().trim();

        System.out.print("Ambang batas: ");
        threshold = scanner.nextDouble();

        System.out.print("Ukuran blok minimum: ");
        min_size = scanner.nextInt();

        scanner.close();
        try {
            img = ImageIO.read(new File(absolute_address_in));
            System.out.println("Gambar berhasil dibaca!");
        } catch (IOException e) {
            System.out.println("Gagal membaca gambar: " + e.getMessage());
        }
    }

    public static double error(int x, int y, int[] size) { //Perhitungan error
        if ("var".equals(error_method)) {
            double var_red = 0;
            double var_green = 0;
            double var_blue = 0;
            Color average_color = get_average_color(x, y, size);

            for (int i = x; i < x + size[0]; i++) {
```

```
            for (int j = y; j < y + size[1]; j++) {
                int val_red = new Color(img.getRGB(j, i)).getRed();
                int val_green = new Color(img.getRGB(j, i)).getGreen();
                int val_blue = new Color(img.getRGB(j, i)).getBlue();
                var_red += Math.pow(val_red - average_color.getRed(), 2);
                var_green += Math.pow(val_green - average_color.getGreen(), 2);
                var_blue += Math.pow(val_blue - average_color.getBlue(), 2);
            }
        }
        double area = size[0] * size[1];
        var_red /= area;
        var_green /= area;
        var_blue /= area;

        double variance = (var_red + var_green + var_blue) / 3;

        return variance;
    } else if ("mad".equals(error_method)) {
        double mad_red = 0;
        double mad_green = 0;
        double mad_blue = 0;
        Color average_color = get_average_color(x, y, size);

        for (int i = x; i < x + size[0]; i++) {
            for (int j = y; j < y + size[1]; j++) {
                int val_red = new Color(img.getRGB(j, i)).getRed();
                int val_green = new Color(img.getRGB(j, i)).getGreen();
                int val_blue = new Color(img.getRGB(j, i)).getBlue();
                mad_red += Math.abs(val_red - average_color.getRed());
                mad_green += Math.abs(val_green - average_color.getGreen());
                mad_blue += Math.abs(val_blue - average_color.getBlue());
            }
        }
        double area = size[0] * size[1];
```

```
            mad_red /= area;
            mad_green /= area;
            mad_blue /= area;

            double mad = (mad_red + mad_green + mad_blue) / 3;

            return mad;
        } else if ("mpd".equals(error_method)) {
            double mpd_red_max = new Color(img.getRGB(y, x)).getRed();;
            double mpd_green_max = new Color(img.getRGB(y, x)).getGreen();
            double mpd_blue_max = new Color(img.getRGB(y, x)).getBlue();

            double mpd_red_min = mpd_red_max;
            double mpd_green_min = mpd_green_max;
            double mpd_blue_min = mpd_blue_max;

            for (int i = x; i < x + size[0]; i++) {
                for (int j = y; j < y + size[1]; j++) {
                    int val_red = new Color(img.getRGB(j, i)).getRed();
                    int val_green = new Color(img.getRGB(j, i)).getGreen();
                    int val_blue = new Color(img.getRGB(j, i)).getBlue();
                    //Max
                    if (mpd_red_max < val_red) {
                        mpd_red_max = val_red;
                    }
                    if (mpd_green_max < val_green) {
                        mpd_green_max = val_green;
                    }
                    if (mpd_blue_max < val_blue) {
                        mpd_blue_max = val_blue;
                    }

                    //Min
```

```
                if (mpd_red_min > val_red) {
                    mpd_red_min = val_red;
                }
                if (mpd_green_min > val_green) {
                    mpd_green_min = val_green;
                }
                if (mpd_blue_min > val_blue) {
                    mpd_blue_min = val_blue;
                }
            }
        }

        double mpd_red = mpd_red_max - mpd_red_min;
        double mpd_green = mpd_green_max - mpd_green_min;
        double mpd_blue = mpd_blue_max - mpd_blue_min;

        double mpd = (mpd_red + mpd_green + mpd_blue) / 3;

        return mpd;
    } else if ("ent".equals(error_method)) {
        int[] red_dist = new int[256];
        int[] green_dist = new int[256];
        int[] blue_dist = new int[256];

        double red_ent = 0;
        double green_ent = 0;
        double blue_ent = 0;

        for (int i = x; i < x + size[0]; i++) {
            for (int j = y; j < y + size[1]; j++) {
                int val_red = new Color(img.getRGB(j, i)).getRed();
                int val_green = new Color(img.getRGB(j, i)).getGreen();
                int val_blue = new Color(img.getRGB(j, i)).getBlue();
```

```
                red_dist[val_red] += 1;
                green_dist[val_green] += 1;
                blue_dist[val_blue] += 1;
            }
        }

        int area = size[0]*size[1];

        for(int i = 0; i <= 255; i++) {
            double prob_red = (double) red_dist[i]/area;
            double prob_green = (double) green_dist[i]/area;
            double prob_blue = (double) blue_dist[i]/area;

            if(prob_red > 0) {
                red_ent += -prob_red*Math.log(prob_red)/Math.log(2);
            }

            if(prob_green > 0) {
                green_ent += -prob_green*Math.log(prob_green)/Math.log(2);
            }

            if(prob_blue > 0) {
                blue_ent += -prob_blue*Math.log(prob_blue)/Math.log(2);
            }

            if(i == 0) {
            }
        }

        double ent = (red_ent + green_ent + blue_ent)/3;
        return ent;
    } else {
        System.out.println("Metode tidak ditemukan");
        System.exit(-1);
```

```
            return -1;
        }
    }

    public static QuadtreeNode block_division(int x, int y, int[] size) { //Algoritma
membagi gambar menjadi 4 & membuat tree
        double variance = error(x, y, size);
        int area = size[0] * size[1];

        if (variance > threshold && area > min_size && area / 4 >= min_size) {
            QuadtreeNode node = new QuadtreeNode(x, y, size, get_average_color(x, y, size),
false);
            int[] new_size_1 = {size[0] / 2, size[1] / 2};
            int[] new_size_2 = {size[0] / 2, size[1] / 2};
            int[] new_size_3 = {size[0] / 2, size[1] / 2};
            int[] new_size_4 = {size[0] / 2, size[1] / 2};

            if (size[0] % 2 != 0 && size[1] % 2 == 0) {
                new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2;
                new_size_3[0] = size[0] / 2 + 1; new_size_3[1] = size[1] / 2;
                new_size_4[0] = size[0] / 2 + 1; new_size_4[1] = size[1] / 2;
            } else if (size[0] % 2 == 0 && size[1] % 2 != 0) {
                new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2 + 1;
                new_size_3[0] = size[0] / 2; new_size_3[1] = size[1] / 2;
                new_size_4[0] = size[0] / 2; new_size_4[1] = size[1] / 2 + 1;
            } else if (size[0] % 2 != 0 && size[1] % 2 != 0) {
                new_size_1[0] = size[0] / 2; new_size_1[1] = size[1] / 2;
                new_size_2[0] = size[0] / 2; new_size_2[1] = size[1] / 2 + 1;
                new_size_3[0] = size[0] / 2 + 1; new_size_3[1] = size[1] / 2;
                new_size_4[0] = size[0] / 2 + 1; new_size_4[1] = size[1] / 2 + 1;
            }

            node.children[0] = block_division(x, y, new_size_1);
```

```
            node.children[1] = block_division(x, y + new_size_1[1], new_size_2);
            node.children[2] = block_division(x + new_size_1[0], y, new_size_3);
            node.children[3] = block_division(x + new_size_1[0], y + new_size_1[1],
new_size_4);

            return node;
        } else {
            return new QuadtreeNode(x, y, size, get_average_color(x, y, size), true);
        }
    }

    public static void save_and_out(BufferedImage img_out) { //Meyimpan hasil gambar
        try {
            File output = new File(absolute_address_out);
            ImageIO.write(img_out, "jpg", output);
            System.out.println("Gambar berhasil dibuat!");
        } catch (IOException e) {
            System.out.println("Gagal menyimpan gambar: " + e.getMessage());
        }
    }

    public static Color get_average_color(int x, int y, int[] size) { //Menghitung rata-rata
warna
        long sum_red = 0;
        long sum_green = 0;
        long sum_blue = 0;

        for (int i = x; i < x + size[0]; i++) {
            for (int j = y; j < y + size[1]; j++) {
                long val_red = new Color(img.getRGB(j, i)).getRed();
                long val_green = new Color(img.getRGB(j, i)).getGreen();
                long val_blue = new Color(img.getRGB(j, i)).getBlue();
                sum_red += val_red;
                sum_green += val_green;
                sum_blue += val_blue;
```

```
            }
        }
        long area = size[0] * size[1];
        long avg_red = sum_red / area;
        long avg_green = sum_green / area;
        long avg_blue = sum_blue / area;
        Color avgColor = new Color((int) avg_red, (int) avg_green, (int) avg_blue);
        return avgColor;
    }

    public static String writeSize(long size) {
        double size_d = size;
        if (size_d < 1024*1024) {
            size_d = size_d / 1024;
            String out = String.format("%.2f", size_d) + " KB";
            return out;
        } else {
            size_d = size_d/(1024*1024);
            String out = String.format("%.2f", size_d) + " MB";
            return out;
        }
    }

    public static BufferedImage make_images(int depth) {

        BufferedImage image = new BufferedImage(img.getWidth(), img.getHeight(),
BufferedImage.TYPE_INT_RGB);
        Graphics2D g = image.createGraphics();
        tree.drawImageByDepth(g, depth);
        g.dispose();

        return image;
    }
}
```

Tangkapan Layar

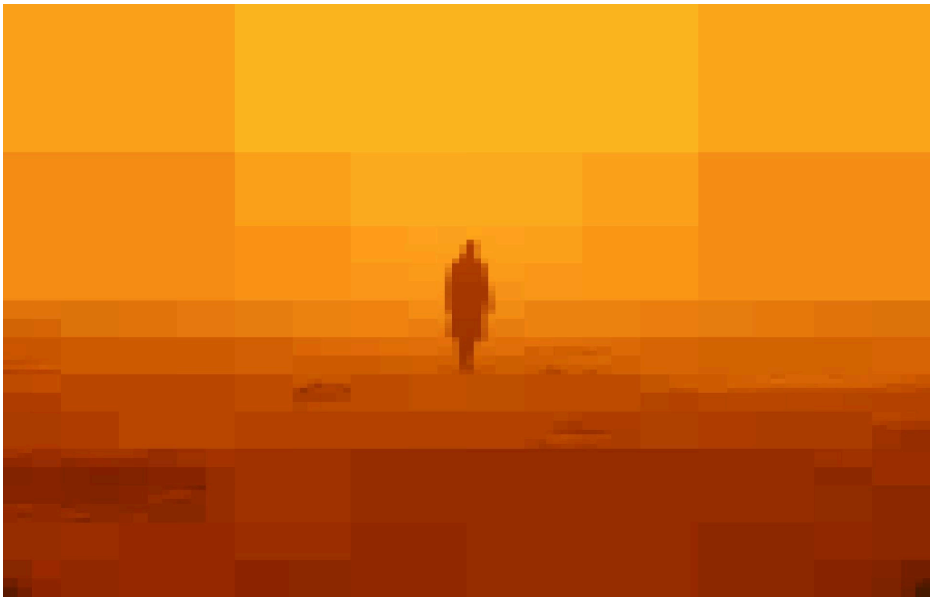
Input Gambar

mono.jpg

color.jpg

1. **Metode variansi pada mono.jpg (threshold = 50, minBlockSize = 50)**

   Gambar Terkompres

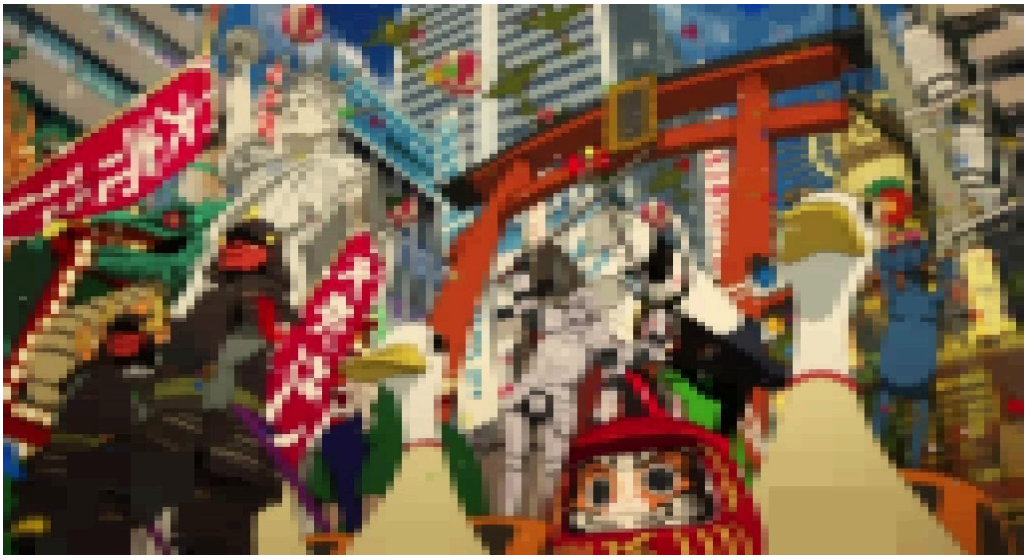


Keluaran Terminal

```
Ukuran gambar input : 140,34 KB
Ukuran gambar output : 52,18 KB
Persentase kompresi : 62.817223 %
Waktu eksekusi: 46007 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 628 simpul
```

2. **Metode variansi gambar pada color.jpg (threshold = 50, minBlockSize = 50)**

Gambar Terkompres

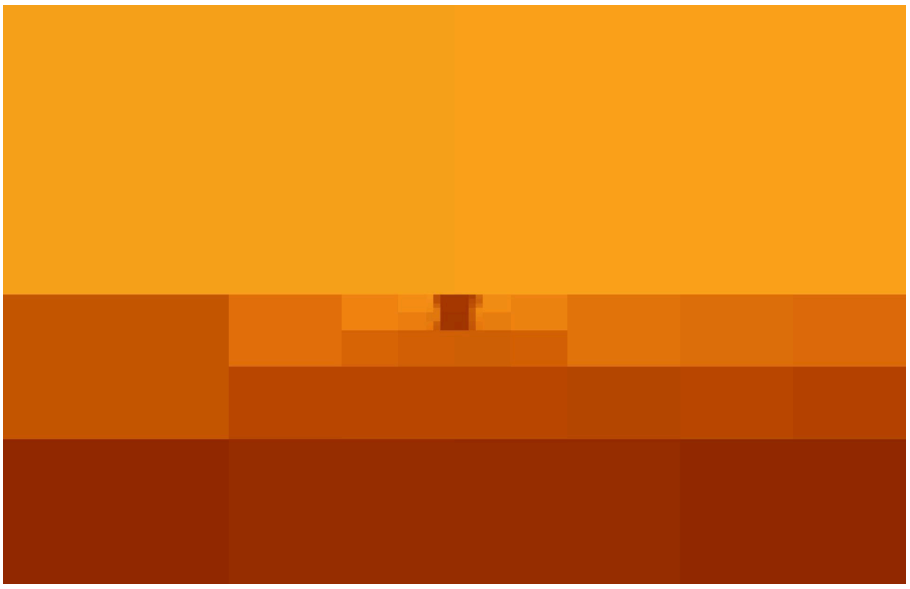


Keluaran Terminal

```
Ukuran gambar input : 190,06 KB
Ukuran gambar output : 126,69 KB
Persentase kompresi : 33.341724 %
Waktu eksekusi: 90450 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 13669 simpul
```

3. **Metode mean absolute deviation pada mono.jpg (threshold = 15, minBlockSize = 50)**

   Gambar Terkompres

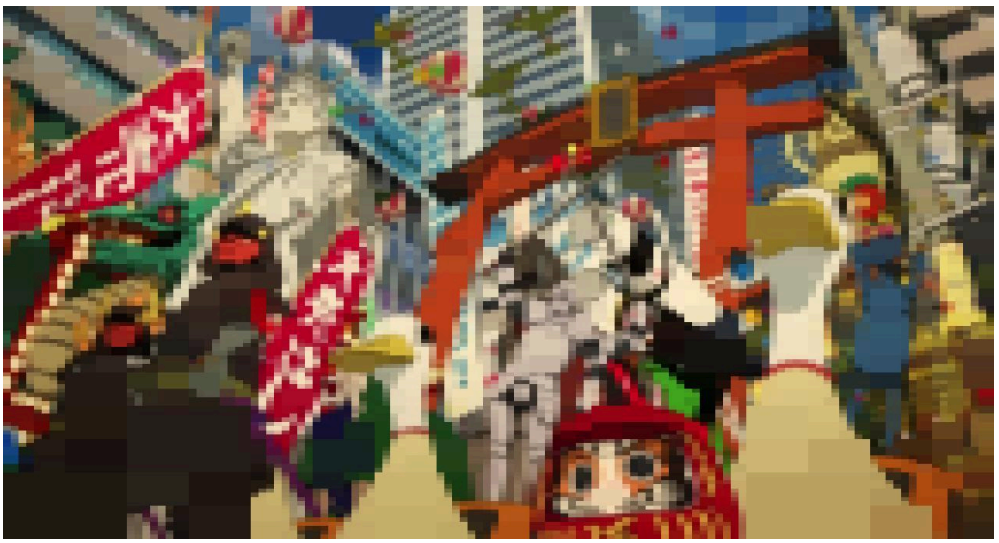


   Keluaran Terminal

```
Ukuran gambar input : 140,34 KB
Ukuran gambar output : 45,22 KB
Persentase kompresi : 67.78142 %
Waktu eksekusi: 19071 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 64 simpul
```

4. **Metode mean absolute deviation pada color.jpg (threshold = 15, minBlockSize = 50)**

   Gambar Terkompres

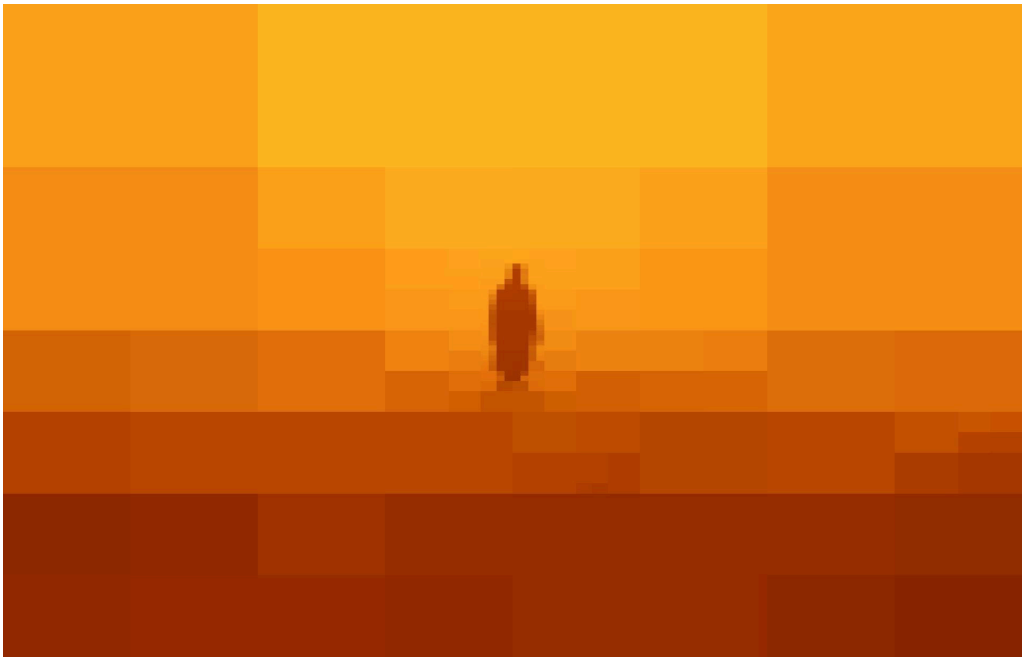


Keluaran Terminal

```
Ukuran gambar input : 190,06 KB
Ukuran gambar output : 118,15 KB
Persentase kompresi : 37.834114 %
Waktu eksekusi: 50581 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 10180 simpul
```

5. **Metode maximum pixel difference pada mono.jpg (threshold = 60, minBlockSize = 60)**

   Gambar Terkompres

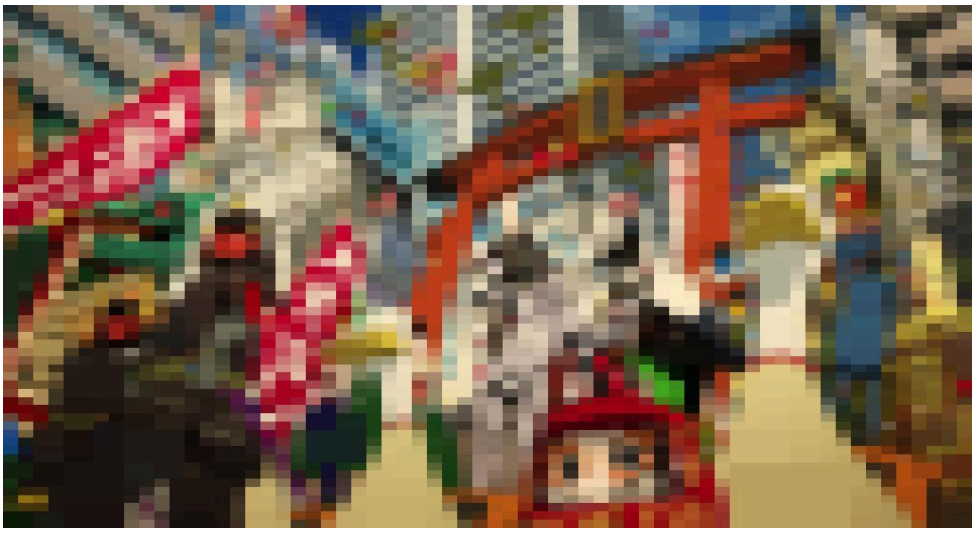


   Keluaran Terminal

```
Ukuran gambar input : 140,34 KB
Ukuran gambar output : 48,42 KB
Persentase kompresi : 65.50181 %
Waktu eksekusi: 29213 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 211 simpul
```

6. **Metode maximum pixel difference pada color.jpg (threshold = 60, minBlockSize = 60)**

   Gambar Terkompres



   Keluaran Terminal

```
Ukuran gambar input : 190,06 KB
Ukuran gambar output : 87,48 KB
Persentase kompresi : 53.970345 %
Waktu eksekusi: 20378 ms
Kedalaman simpul dalam implementasi : 7 simpul
Banyak simpul dalam implementasi : 3703 simpul
```

7. **Metode entropi pada mono.jpg (threshold = 2, minBlockSize = 50)**

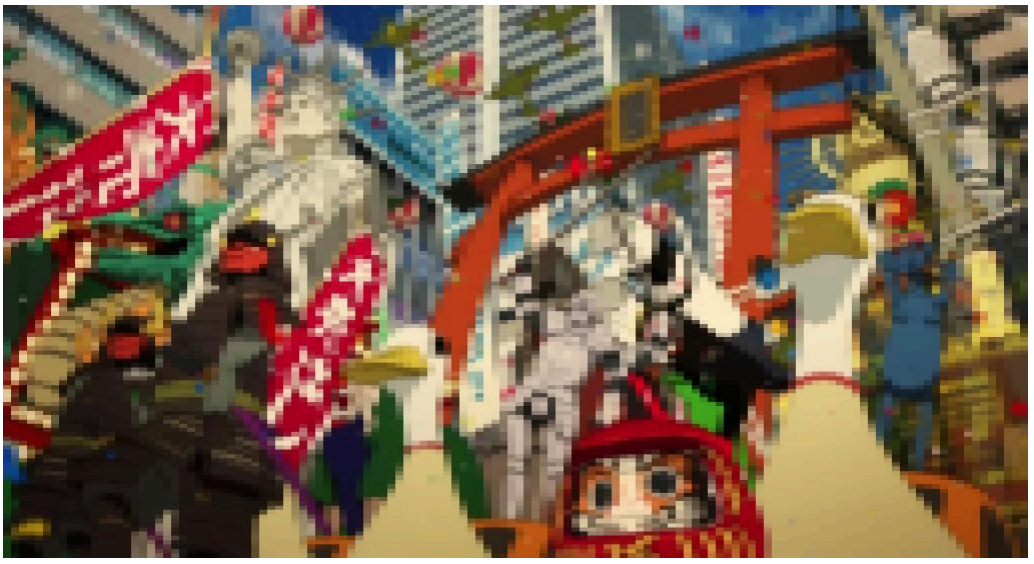
   Gambar Terkompres

Keluaran Terminal

```
Ukuran gambar input : 140,34 KB
Ukuran gambar output : 68,52 KB
Persentase kompresi : 51.175644 %
Waktu eksekusi: 27606 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 10069 simpul
```

8. **Metode entropi pada color.jpg (threshold = 2, minBlockSize = 50)**

Gambar Terkompres



Keluaran Terminal

```
Ukuran gambar input : 190,06 KB
Ukuran gambar output : 127,92 KB
Persentase kompresi : 32.691734 %
Waktu eksekusi: 26565 ms
Kedalaman simpul dalam implementasi : 8 simpul
Banyak simpul dalam implementasi : 15631 simpul
```

Analisis

- Parameter ambang batas adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut. Dengan demikian, semakin kecil nilai ambang batas, nilai persentase kompresi yang dihasilkan akan semakin kecil.
- Gambar yang memiliki sedikit warna akan lebih mudah dikompresi oleh algoritma yang dibuat dibandingkan dengan gambar yang memiliki lebih banyak warna.
- Parameter ukuran blok mengatur ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Dengan demikian, semakin besar nilai parameter ini, nilai persentase kompresi akan semakin besar.
- Kompleksita algoritma : Kita dapat menyusun relasi rekurensinya sebagai

$$T(m,n) = \begin{cases} 4T(\frac{m}{2}, \frac{n}{2}) + cmn, & mn \geq k \\ mn, & mn < k \end{cases}$$

Kita dapat anggap penghentian algoritma karena nilai thresholdnya sebagai peristiwa yang jarang terjadi. Karena lebih kemungkinan warna pada suatu gambar lebih bervariasi daripada tidak bervariasi

Perhatikan bahwa kita melakukan pembagian sampai level $log_4(mn/k) = O(logmn)$ dan pada level ke-i, kita mengerjakan 4^i subproblem dengan kompleksitas $O(mn/4^i)$. Jadi, pada tiap level, kita mengerjakan perhitungan dengan kompleksitas $O(\frac{mn}{4^i} 4^i) = O(mn)$.

Jadi, kompleksitas waktu terburuk untuk keseluruhan algoritma adalah $O(mn \cdot log(mn))$

Pranala

Kode dapat diakses di : https://github.com/countz-zero/Tucil2_10122010_10122074

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	☑	☐