IF2211 Strategi Algoritma

Kompresi Gambar dengan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2 Tahun Akademik 2024/2025

Disusun oleh:

13523135 - Ahmad Syafiq

PROGRAM STUDI TEKNIK INFORMATIKA

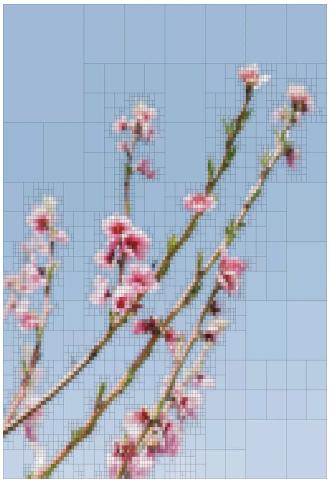
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

2024

DAFTAR ISI

BAB I
DESKRIPSI MASALAH



Gambar 1.1.
Quadtree dalam kompresi gambar
(Sumber: https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00)

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

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

Dalam tugas kecil ini, penulis ditugaskan untuk membuat program sederhana dalam bahasa C/C#/C++/Java (CLI) yang mengimplementasikan algoritma divide and conquer untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan parameter yaitu metode pengukuran galat, ambang batas, ukuran blok minimum, dan persentase kompresi sebagai masukan pengguna.

Alur program yang ditugaskan:

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi (floating number, 1.0 = 100%), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [INPUT] alamat absolut gif (bonus).
8. [OUTPUT] waktu eksekusi.
9. [OUTPUT] ukuran gambar sebelum.
10. [OUTPUT] ukuran gambar setelah.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon.
13. [OUTPUT] banyak simpul pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

BAB II
TEORI SINGKAT

2.1. Algoritma Divide and Conquer

Algoritma divide and conquer merupakan pendekatan yang digunakan dalam pemecahan masalah yang umumnya dengan langkah divide, conquer, kemudian combine. Divide artinya membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil. Idealnya setiap upa-persoalan berukuran hampir sama. Conquer artinya menyelesaikan (solve) setiap upa-persoalan. Penyelesaian persoalan bisa dilakukan secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar. Kemudian combine yang artinya menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Algoritma ini cocok digunakan untuk menyelesaikan masalah yang dapat dibagi menjadi beberapa bagian kecil dengan karakteristik yang sama. Meskipun pada awalnya mungkin terlihat rumit, penggunaan algoritma divide and conquer memungkinkan penyelesaian masalah yang lebih efisien karena memecah masalah menjadi bagian-bagian yang lebih kecil dan lebih mudah dipecahkan. Algoritma divide and conquer memungkinkan pemecahan masalah yang lebih efektif dan efisien dalam berbagai konteks, seperti pemrosesan paralel, pengurutan, dan pencarian. Dengan demikian kompleksitas waktu untuk algoritma divide and conquer umumnya adalah sebagai berikut

$$T(n) = \begin{cases} g(n) & , n \leq \mathrm{n}_0 \\ T(n_1) + T(n_2) \ldots + T(n_r) + f(n) & , n > n_0 \end{cases}$$

dengan

- $T(n)$: kompleksitas waktu penyelesaian persoalan P yang berukuran n
- $g(n)$: kompleksitas waktu untuk SOLVE jika n sudah berukuran kecil
- $T(n_1) + T(n_2) \ldots + T(n_r)$: kompleksitas waktu untuk memproses setiap upa-persoalan
- $f(n)$: kompleksitas waktu untuk COMBINE solusi dari masing-masing upa-persoalan
- Tahap DIVIDE dapat dilakukan dalam O(1), sehingga tidak dimasukkan ke dalam formula

2.2. Quadtree

Quadtree adalah struktur data berbasis pohon di mana setiap simpul internal memiliki tepat empat anak. Struktur ini merupakan versi dua dimensi dari octree dan sering digunakan untuk membagi ruang dua dimensi secara rekursif menjadi empat bagian atau kuadran. Data yang tersimpan di setiap sel daun bisa bervariasi tergantung aplikasinya, tetapi umumnya sel

daun merepresentasikan unit informasi spasial yang penting. Wilayah yang dibagi bisa berbentuk persegi, persegi panjang, atau bentuk lain yang lebih fleksibel.

Beberapa karakteristik umum dari quadtree:

- Membagi ruang menjadi sel-sel yang bisa beradaptasi dengan kebutuhan data.
- Setiap sel memiliki kapasitas maksimum. Jika kapasitas ini tercapai, sel akan terbagi lebih lanjut.
- Struktur pohonnya mencerminkan pembagian ruang yang dilakukan oleh quadtree.

BAB III
METODE PENYELESAIAN

Algoritma divide and conquer dapat diterapkan dalam proses kompresi gambar menggunakan metode quadtree. Pertama, gambar dianggap sebagai array dua dimensi yang menyimpan nilai warna merah, hijau, dan biru untuk setiap pikselnya. Kompresi gambar dengan quadtree dilakukan dengan merepresentasikan satu blok yang seragam dengan satu warna. Ide utama dalam kompresi gambar dengan quadtree adalah membagi gambar secara rekursif ke dalam bagian-bagian yang lebih kecil hingga memenuhi tingkat keseragaman tertentu. Prinsip divide and conquer dalam metode ini dapat dijelaskan sebagai berikut:

1. Divide

   Persoalan dalam mengkompresi gambar dapat dibagi menjadi upa-persoalan yang lebih kecil. Hal tersebut terjadi jika blok belum bisa direpresentasikan dengan satu warna. Pembagian dilakukan dengan membagi rata blok menjadi empat blok yang lebih kecil, yaitu kiri-atas, kanan-atas, kiri-bawah, dan kanan-bawah, kemudian menyelesaikan secara rekursif upa-persoalan subblok.

2. Conquer

   Pada setiap langkah rekursi, setiap upa-persoalan akan diselesaikan secara terpisah. Jika kondisi memenuhi—dalam hal ini, galat dari warna rata-rata dari blok terhadap warna asli blok berada di bawah ambang batas—warna blok tersebut akan disamakan menjadi warna rata-rata. Pembagian dihentikan, upa-persoalan dianggap sebagai daun, dan warna rata-rata akan digunakan dalam rekonstruksi gambar.

3. Combine

   Setelah menyelesaikan upa-persoalan dari langkah conquer, maka warna yang dapat merepresentasikan blok sesuai ketentuan diperoleh. Untuk mempermudah pemrosesan, rekonstruksi gambar dilakukan dengan menelusuri pohon yang sudah jadi, kemudian mengisi nilai pada array dua dimensi dengan nilai warna dari daun yang berkorespondensi.

BAB IV
IMPLEMENTASI

4.1. Implementasi Program

Program diimplementasikan dengan bahasa Java dengan satu paket berisi file Main.java, Quadtree.java, dan Utils.java. Adapun detail terkait tiap file adalah sebagai berikut:

4.1.1. Quadtree.java

File ini berisi implementasi kelas quadtree serta kelas node. Konstruktor quadtree berisi algoritma utama mengkompres gambar secara divide and conquer.

4.1.1.1. Atribut Node

Nama	Tipe	Keterangan
x	integer	absis titik terkiri dari blok yang dicakup
y	integer	ordinat titik teratas dari blok yang dicakup
r	integer	rata-rata nilai warna merah dari blok yang dicakup
g	integer	rata-rata nilai warna hijau dari blok yang dicakup
b	integer	rata-rata nilai warna biru dari blok yang dicakup

4.1.1.3. Atribut Quadtree

Nama	Tipe	Keterangan
width	integer	lebar blok gambar yang dicakup
height	integer	tinggi blok gambar yang dicakup
root	Node	data akar pohon
depth	integer	kedalaman pohon terhadap pohon utama (level). pohon utama memiliki kedalaman 1
nw	Quadtree	upapohon yang merepresentasikan bagian kiri atas
ne	Quadtree	upapohon yang merepresentasikan bagian kanan atas
sw	Quadtree	upapohon yang merepresentasikan bagian kiri bawah
se	Quadtree	upapohon yang merepresentasikan bagian kanan bawah
isLeaf	boolean	benar jika dan hanya jika simpul ini adalah daun

4.1.1.4. Metode Quadtree

Fungsi	Keterangan
`Quadtree`	konstruktor
`maxDepth`	mengembalikan kedalaman maksimum
`leafCount`	mengembalikan jumlah daun
`computeAverageColor`	menghitung warna rata-rata dari blok gambar yang dicakup
`shouldSplit`	helper function untuk menentukan apakah masih perlu memecah blok menjadi 4 dalam konstruksi quadtree berdasarkan parameter
`computeError`	helper function untuk menghitung galat blok yang dicakup dalam menentukan apakah perlu memecah blok
`computeVariance`	helper function untuk menghitung galat berdasarkan variansi
`computeMAD`	helper function untuk menghitung galat berdasarkan Mean Absolute Difference
`computeMPD`	helper function untuk menghitung galat berdasarkan Max Pixel Difference
`computeEntropy`	helper function untuk menghitung galat berdasarkan entropi
`computeChannelEntropy`	helper function untuk menghitung entropi tiap kanal warna
`computeSSIM`	helper function untuk menghitung galat berdasarkan Structural Similarity Index (**bonus**)
`computeChannelSSIM`	helper function untuk menghitung Structural Similarity Index tiap kanal warna (**bonus**)
`getX`	mengembalikan nilai x dari akar pohon
`getY`	mengembalikan nilai y dari akar pohon
`getRed`	mengembalikan nilai warna merah dari akar pohon
`getGreen`	mengembalikan nilai warna hijau dari akar pohon
`getBlue`	mengembalikan nilai warna biru dari akar pohon

4.1.2. Utils.java

File ini berisi metode-metode statis yang membantu proses kompresi gambar, terutama pemuatan gambar, penciptaan gambar, serta penciptaan GIF.

4.1.2.1. Metode

Nama	Keterangan
`pathToArray`	menerima path absolut sebuah gambar (berekstensi jpg, jpeg, atau png), mengembalikan array dua dimensi berisi [r,g,b]
`saveQuadtreeAsImage`	menyimpan quadtree sebagai gambar di path absolut
`fillImageFromQuadtree`	helper function membuat gambar dari quadtree
`getFileSize`	mengembalikan ukuran file
`generateQuadtreeFrames`	menyusun senarai gambar untuk GIF (**bonus)**
`fillQuadtreeFrame`	helper function untuk membuat gambar dari quadtree (**bonus)**
`createGIF`	Menyimpan senarai gambar menjadi gif di path absolut (**bonus)**

4.1.2. Utils.java

File ini digunakan untuk menjalankan aplikasi program. File ini berisi implementasi menu utama.

4.1.2.1. Metode

Nama	Keterangan
`main`	entry point program
`findOptimalThreshold`	fungsi kompresi gambar berkali-kali dengan penyesuaian threshold secara binary search untuk mendapatkan target kompresi (**bonus**)

4.2. Kode Sumber

4.2.1. Quadtree.java

```
import java.util.HashMap;
import java.util.Map;

public class Quadtree {
   int width, height;
   Node root;
   int depth; // root = 1
   Quadtree nw, ne, sw, se;
   boolean isLeaf;

   public Quadtree(int[][][] imageArray, int x, int y, int width, int height,
double threshold, int minBlockSize, int errorMethod, int depth) {
       if (width*height == 0) {
           isLeaf = false;
```

```
            return;
        }
        this.width = width;
        this.height = height;
        int[] avgColor = computeAverageColor(imageArray, x, y, width, height);
        this.root = new Node(x, y, avgColor[0], avgColor[1], avgColor[2]);
        this.depth = depth;
        this.isLeaf = true;
        if (shouldSplit(imageArray, avgColor, x, y, width, height, threshold,
minBlockSize, errorMethod)) {
            this.isLeaf = false;
            int westW = width / 2;
            int northH = height / 2;

            int eastW = width - westW;
            int southH = height - northH;
            nw = new Quadtree(imageArray, x, y, westW, northH, threshold,
minBlockSize, errorMethod, depth+1);
            ne = new Quadtree(imageArray, x + westW, y, eastW, northH,
threshold, minBlockSize, errorMethod, depth+1);
            sw = new Quadtree(imageArray, x, y + northH, westW, southH,
threshold, minBlockSize, errorMethod, depth+1);
            se = new Quadtree(imageArray, x + westW, y + northH, eastW, southH,
threshold, minBlockSize, errorMethod, depth+1);
        }
    }

    public int maxDepth() {
        if (isLeaf) {
            return 1;
        }
        int nwDepth = (nw != null) ? nw.maxDepth() : 0;
        int neDepth = (ne != null) ? ne.maxDepth() : 0;
        int swDepth = (sw != null) ? sw.maxDepth() : 0;
        int seDepth = (se != null) ? se.maxDepth() : 0;

        return 1 + Math.max(Math.max(nwDepth, neDepth), Math.max(swDepth,
seDepth));
    }

    public int leafCount() {
        if (isLeaf) {
            return 1;
        }
        int nwCount = (nw != null) ? nw.leafCount() : 0;
        int neCount = (ne != null) ? ne.leafCount() : 0;
        int swCount = (sw != null) ? sw.leafCount() : 0;
        int seCount = (se != null) ? se.leafCount() : 0;

        return 1 + nwCount + neCount + swCount + seCount;
    }

    private int[] computeAverageColor(int[][][] image, int x, int y, int width,
int height) {
        int sumR = 0, sumG = 0, sumB = 0, count = width * height;
```

```
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                sumR += image[i][j][0];
                sumG += image[i][j][1];
                sumB += image[i][j][2];
            }
        }
        return new int[] {sumR / count, sumG / count, sumB / count};
    }

    private boolean shouldSplit(int[][][] image, int[] avgColor, int x, int y,
int width, int height, double threshold, int minBlockSize, int errorMethod) {

        if (width*height/4 < minBlockSize) return false;

        double error = computeError(image, avgColor, x, y, width, height,
errorMethod);
        if (errorMethod == 5) {
            return error < threshold;
        }
        return error > threshold;
    }

    private double computeError(int[][][] image, int[] avgColor, int x, int y,
int width, int height, int errorMethod) {
        double error = 0;
        if (errorMethod == 1) {
            error = computeVariance(image, avgColor, x, y, width, height);
        }
        else if (errorMethod == 2) {
            error = computeMAD(image, avgColor, x, y, width, height);
        }
        else if (errorMethod == 3) {
            error = computeMPD(image, x, y, width, height);
        }
        else if (errorMethod == 4) {
            error = computeEntropy(image, x, y, width, height);
        }
        else if (errorMethod == 5) {
            error = computeSSIM(image, avgColor, x, y, width, height);
        }

        return error;
    }


    // VARIANCE
    private double computeVariance(int[][][] image, int[] avgColor, int x, int
y, int width, int height) {
        double redVariance = 0;
        double greenVariance = 0;
        double blueVariance = 0;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                redVariance += Math.pow(image[i][j][0] - avgColor[0], 2);
```

```
                greenVariance += Math.pow(image[i][j][1] - avgColor[1], 2);
                blueVariance += Math.pow(image[i][j][2] - avgColor[2], 2);
            }
        }
        redVariance /= width*height;
        greenVariance /= width*height;
        blueVariance /= width*height;
        return (redVariance + greenVariance + blueVariance) / 3;
    }

    // MEAN ABSOLUTE DEVIATION (MAD)
    private double computeMAD(int[][][] image, int[] avgColor, int x, int y,
int width, int height) {
        double redMAD = 0;
        double greenMAD = 0;
        double blueMAD = 0;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                redMAD += Math.abs(image[i][j][0] - avgColor[0]);
                greenMAD += Math.abs(image[i][j][1] - avgColor[1]);
                blueMAD += Math.abs(image[i][j][2] - avgColor[2]);
            }
        }
        redMAD /= width*height;
        greenMAD /= width*height;
        blueMAD /= width*height;
        return (redMAD + greenMAD + blueMAD) / 3;
    }

    // MAX PIXEL DIFFERENCE (MPD)
    private double computeMPD(int[][][] image, int x, int y, int width, int
height) {
        double redMax = image[y][x][0];
        double redMin = image[y][x][0];
        double greenMax = image[y][x][1];
        double greenMin = image[y][x][1];
        double blueMax = image[y][x][2];
        double blueMin = image[y][x][2];

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                if (image[i][j][0] > redMax) redMax = image[i][j][0];
                if (image[i][j][0] < redMin) redMin = image[i][j][0];

                if (image[i][j][1] > greenMax) greenMax = image[i][j][1];
                if (image[i][j][1] < greenMin) greenMin = image[i][j][1];

                if (image[i][j][2] > blueMax) blueMax = image[i][j][2];
                if (image[i][j][2] < blueMin) blueMin = image[i][j][2];
            }
        }
        return ((redMax - redMin) + (greenMax - greenMin) + (blueMax -
blueMin)) / 3;
    }
```

```
   // ENTROPY
   private double computeEntropy(int[][][] image, int x, int y, int width, int
height) {
       return (computeChannelEntropy(image, x, y, width, height, 0) +
               computeChannelEntropy(image, x, y, width, height, 1) +
               computeChannelEntropy(image, x, y, width, height, 2)) / 3.0;
   }

   private double computeChannelEntropy(int[][][] image, int x, int y, int
width, int height, int channel) {
       Map<Integer, Integer> frequencyMap = new HashMap<>();
       int totalPixels = width * height;

       for (int i = y; i < y + height; i++) {
           for (int j = x; j < x + width; j++) {
               int pixelValue = image[i][j][channel];
               frequencyMap.put(pixelValue,
frequencyMap.getOrDefault(pixelValue, 0) + 1);
           }
       }

       double entropy = 0.0;
       for (int count : frequencyMap.values()) {
           double probability = (double) count / totalPixels;
           entropy -= probability * (Math.log(probability) / Math.log(2));
       }
       return entropy;
   }

   private double computeSSIM(int[][][] image, int[] avgColor, int x, int y,
int width, int height) {
       double ssimR = computeChannelSSIM(image, avgColor[0], x, y, width,
height, 0);
       double ssimG = computeChannelSSIM(image, avgColor[1], x, y, width,
height, 1);
       double ssimB = computeChannelSSIM(image, avgColor[2], x, y, width,
height, 2);

       return 0.2989*ssimR + 0.5870*ssimG + 0.1140*ssimB;
   }

   private double computeChannelSSIM(int[][][] image, int mean, int x, int y,
int width, int height, int channel) {
       final double C2 = 58.5225;
       double varianceX = 0;
       for (int i = y; i < y + height; i++) {
           for (int j = x; j < x + width; j++) {
               double diffX = image[i][j][channel] - mean;
               varianceX += diffX * diffX;
           }
       }
       varianceX /= (width * height);

       //penyederhanaan karena varianceY = 0, cov = 0, meanX=meanY=mean
       return C2 / (varianceX + C2);
```

```
    }


        private static class Node {
        int r, g, b, x, y;

        public Node(int x, int y, int r, int g, int b) {
            this.x = x;
            this.y = y;
            this.r = r;
            this.g = g;
            this.b = b;
        }
    }

    public int getX() {
        return this.root.x;
    }
    public int getY() {
        return this.root.y;
    }
    public int getRed() {
        return this.root.r;
    }
    public int getGreen() {
        return this.root.g;
    }
    public int getBlue() {
        return this.root.b;
    }
}
```

4.2.2. Utils.java

```
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileOutputStream;
import java.io.IOException;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import com.github.dragon66.AnimatedGIFWriter;
import java.io.OutputStream;


public class Utils {

    public static int[][][] pathToArray(String imagePath) {
        try {
            BufferedImage image = ImageIO.read(new File(imagePath));
            int width = image.getWidth();
            int height = image.getHeight();
            int[][][] imageArray = new int[height][width][3];
```

```
            for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                    int rgb = image.getRGB(x, y);
                    imageArray[y][x][0] = (rgb >> 16) & 0xFF;
                    imageArray[y][x][1] = (rgb >> 8) & 0xFF;
                    imageArray[y][x][2] = rgb & 0xFF;
                }
            }
            return imageArray;
        } catch (Exception e) {
            return null;
        }
    }

    public static void saveQuadtreeAsImage(Quadtree tree, int width, int
height, String outputPath) throws Exception {
        BufferedImage img = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        String outputExt = outputPath.substring(outputPath.lastIndexOf('.') +
1).toLowerCase();
        fillImageFromQuadtree(tree, img);
        ImageIO.write(img, outputExt, new File(outputPath));
    }

    public static void fillImageFromQuadtree(Quadtree tree, BufferedImage img)
{
        if (tree.root == null) return;

        if (tree.isLeaf) {
            for (int i = tree.getY(); i < tree.getY() + tree.height; i++) {
                for (int j = tree.getX(); j < tree.getX() + tree.width; j++) {
                    int rgb = (tree.getRed() << 16) | (tree.getGreen() << 8) |
tree.getBlue();
                    img.setRGB(j, i, rgb);
                }
            }
        } else {
            fillImageFromQuadtree(tree.nw, img);
            fillImageFromQuadtree(tree.ne, img);
            fillImageFromQuadtree(tree.sw, img);
            fillImageFromQuadtree(tree.se, img);
        }
    }

    public static long getFileSize(String filePath) {
        File file = new File(filePath);
        return file.length();
    }

    public static List<BufferedImage> generateQuadtreeFrames(Quadtree tree, int
width, int height, int maxDepth) {
        List<BufferedImage> frames = new ArrayList<>();
        int step = (maxDepth - 1) / 4;
        if (step < 1) step = 1;

        for (int d = 1; d < maxDepth; d += step) {
```

```
            BufferedImage img = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
            fillQuadtreeFrame(tree, img, d, 1);
            frames.add(img);
        }
        BufferedImage img = new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB);
        fillQuadtreeFrame(tree, img, maxDepth, 1);
        frames.add(img);

        return frames;
    }

    private static void fillQuadtreeFrame(Quadtree tree, BufferedImage img, int
targetDepth, int currentDepth) {
        if (tree == null || tree.root == null) return;

        if (currentDepth > targetDepth) return;

        if (tree.isLeaf || currentDepth == targetDepth) {
            for (int i = tree.getY(); i < tree.getY() + tree.height; i++) {
                for (int j = tree.getX(); j < tree.getX() + tree.width; j++) {
                    int rgb = (tree.getRed() << 16) | (tree.getGreen() << 8) |
tree.getBlue();
                    img.setRGB(j, i, rgb);
                }
            }
        } else {
            fillQuadtreeFrame(tree.nw, img, targetDepth, currentDepth + 1);
            fillQuadtreeFrame(tree.ne, img, targetDepth, currentDepth + 1);
            fillQuadtreeFrame(tree.sw, img, targetDepth, currentDepth + 1);
            fillQuadtreeFrame(tree.se, img, targetDepth, currentDepth + 1);
        }
    }


    public static void createGIF(List<BufferedImage> frames, String outputPath,
int delay, boolean loop) throws IOException {
        if (frames.isEmpty()) {
            throw new IllegalArgumentException("No frames to create GIF");
        }

        File outputFile = new File(outputPath);
        try (OutputStream outputStream = new FileOutputStream(outputFile)) {
            AnimatedGIFWriter writer = new AnimatedGIFWriter(false);
            writer.prepareForWrite(outputStream, -1, -1);

            for (BufferedImage frame : frames) {
                writer.writeFrame(outputStream, frame, delay);
            }

            writer.finishWrite(outputStream);
        } catch (Exception e) {
            throw new RuntimeException(e);
        }
    }
```

```
}
```

4.2.3. Main.java

```
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.IOException;
import java.util.List;
import java.util.Scanner;
import java.io.File;


public class Main {
   static class BinSearchResult {
       double threshold;
       Quadtree tree;

       BinSearchResult(double threshold, Quadtree tree) {
           this.threshold = threshold;
           this.tree = tree;
       }
   }
   public static void main(String[] args) {
       Scanner scanner = new Scanner(System.in);


       try {
           // 1. [INPUT] Alamat absolut gambar yang akan dikompresi
           String inputPath;
           do {
               System.out.print("Masukkan path gambar input: ");
               inputPath = scanner.nextLine().trim();
               File file = new File(inputPath);

               if (!file.exists() || !file.isFile()) {
                   System.out.println("Error: File tidak ditemukan. Coba
lagi.\n");
                   continue;
               }
               String lowerPath = inputPath.toLowerCase();
               if (!lowerPath.endsWith(".jpg") && !lowerPath.endsWith(".jpeg")
&&
                       !lowerPath.endsWith(".png")) {
                   System.out.println("Error: Format file tidak didukung.
Harus berupa .jpg, .jpeg, atau .png. Coba lagi.\n");
                   continue;
               }

               try {
                   BufferedImage testImage = ImageIO.read(file);
                   if (testImage == null) {
                       System.out.println("Error: Tidak bisa membaca gambar.
```

```
Coba lagi.\n");
                        continue;
                    }
                } catch (IOException e) {
                    System.out.println("Error: Tidak bisa membaca file. Coba
lagi.\n");
                    continue;
                }

                break;

            } while (true);


            // 2. [INPUT] Metode perhitungan error (gunakan penomoran sebagai
input)
            int errorMethod;
            do {
                System.out.println("Pilih metode perhitungan error:");
                System.out.println("1. Variance\n2. Mean Absolute Deviation
(MAD)\n3. Max Pixel Difference\n4. Entropy\n5. Structural Similarity Index
(SSIM)");
                System.out.print("Masukkan nomor metode: ");
                while (!scanner.hasNextInt()) {
                    System.out.println("Error: Input harus berupa angka.");
                    scanner.next();
                }
                errorMethod = scanner.nextInt();
                if (errorMethod < 1 || errorMethod > 5) {
                    System.out.println("Error: Pilihan metode tidak valid. Coba
lagi.\n");
                }
            } while (errorMethod < 1 || errorMethod > 5);



            // 3. [INPUT] Ambang batas (threshold)
            double threshold;
            do {
                System.out.print("Masukkan ambang batas (threshold): ");
                while (!scanner.hasNextDouble()) {
                    System.out.println("Error: Input harus berupa angka.");
                    scanner.next();
                }
                threshold = scanner.nextDouble();
                if (threshold < 0) {
                    System.out.println("Error: Threshold harus lebih besar atau
sama dengan 0. Coba lagi.\n");
                }
                else if (errorMethod == 5 && (threshold <= 0 || threshold >=
1)) {
                    System.out.println("Error: Threshold untuk SSIM harus dalam
rentang (0,1). Coba lagi.\n");
                }
            } while ((errorMethod == 5 && (threshold <= 0 || threshold >= 1))
|| threshold < 0);
```

```
            // 4. [INPUT] Ukuran blok minimum
            int minBlockSize;
            do {
                System.out.print("Masukkan ukuran blok minimum: ");
                while (!scanner.hasNextInt()) {
                    System.out.println("Error: Input harus berupa angka.");
                    scanner.next();
                }
                minBlockSize = scanner.nextInt();
                if (minBlockSize < 1) {
                    System.out.println("Error: Ukuran blok minimum harus lebih
besar dari 0. Coba lagi.\n");
                }
            } while (minBlockSize < 1);



            // 5. [INPUT] Target persentase kompresi
            double targetCompression;
            do {
                System.out.print("Masukkan target persentase kompresi (0 untuk
menonaktifkan mode ini): ");
                while (!scanner.hasNextDouble()) {
                    System.out.println("Error: Input harus berupa angka.");
                    scanner.next();
                }
                targetCompression = scanner.nextDouble();
                if (targetCompression < 0 || targetCompression > 1) {
                    System.out.println("Error: Persentase kompresi harus di
antara 0 dan 1. Coba lagi.\n");
                }
            } while (targetCompression < 0 || targetCompression > 1);



            // 6. [INPUT] Alamat absolut gambar hasil kompresi
            String outputPath;
            do {
                System.out.print("Masukkan path gambar output: ");
                outputPath = scanner.nextLine().trim();

                File outputFile = new File(outputPath);
                File outputDir = outputFile.getParentFile();

                if (outputPath.isEmpty()) {
                    continue;
                }

                if (!outputPath.contains(".")) {
                    System.out.println("Error: Path harus memiliki ekstensi
file. Coba lagi.\n");
                    continue;
                }
```

```
                String inputExt =
inputPath.substring(inputPath.lastIndexOf('.') + 1).toLowerCase();
                String outputExt =
outputPath.substring(outputPath.lastIndexOf('.') + 1).toLowerCase();
                if (!inputExt.equals(outputExt)) {
                    System.out.println("Error: Ekstensi file input dan output
harus sama. Coba lagi.\n");
                    continue;
                }

                if (outputDir != null && !outputDir.exists()) {
                    System.out.println("Error: Folder tujuan tidak ditemukan.
Coba lagi.\n");
                    continue;
                }

                break;

            } while (true);

            File inputFile = new File(inputPath);
            int[][][] imageArray = Utils.pathToArray(inputPath);

            // 7. [INPUT] Alamat absolut GIF
            String gifOutputPath;
            boolean generateGif;
            do {
                System.out.print("Masukkan path GIF output (kosongkan jika
tidak ingin menyimpan GIF): ");
                gifOutputPath = scanner.nextLine().trim();

                if (gifOutputPath.isEmpty()) {
                    generateGif = false;
                    break;
                }


                File gifFile = new File(gifOutputPath);
                String gifExt =
gifOutputPath.substring(gifOutputPath.lastIndexOf('.') + 1).toLowerCase();
                if (!gifExt.equals("gif")) {
                    System.out.println("Error: Ekstensi file harus .gif. Coba
lagi.\n");
                    continue;
                }

                File gifDir = gifFile.getParentFile();
                if (gifDir != null && (!gifDir.exists() ||
!gifDir.isDirectory())) {
                    System.out.println("Error: Folder tujuan tidak ditemukan.
Coba lagi.\n");
                    continue;
                }

                generateGif = true;
```

```
                break;
            } while (true);


            //START
            long startTime = System.nanoTime();

            // COMPRESSION
            Quadtree tree;
            if (targetCompression > 0) {
                BinSearchResult binSearchResult =
findOptimalThreshold(imageArray, inputFile, outputPath, targetCompression,
errorMethod);
                threshold = binSearchResult.threshold;
                tree = binSearchResult.tree;
            }
            else {
                tree = new Quadtree(imageArray, 0, 0, imageArray[0].length,
imageArray.length, threshold, minBlockSize, errorMethod, 1);
            }

            Utils.saveQuadtreeAsImage(tree, imageArray[0].length,
imageArray.length, outputPath);
            if (generateGif) {
                int maxDepth = tree.maxDepth();
                List<BufferedImage> frames =
Utils.generateQuadtreeFrames(tree, imageArray[0].length, imageArray.length,
maxDepth);
                Utils.createGIF(frames, gifOutputPath, 500, true);
            }
            long endTime = System.nanoTime();

            System.out.println("Berhasil Kompresi!");
//              8. [OUTPUT] waktu eksekusi
            System.out.println("1. Waktu eksekusi: "+ (endTime -
startTime)/1_000_000 + " ms");

//              9. [OUTPUT] ukuran gambar sebelum
            System.out.println("2. Sebelum: " + String.format("%.2f KB",
Utils.getFileSize(inputPath) / 1024.0));

//              10. [OUTPUT] ukuran gambar setelah
            System.out.println("3. Sesudah: " + String.format("%.2f KB",
Utils.getFileSize(outputPath) / 1024.0));

//              11. [OUTPUT] persentase kompresi
            System.out.println("4. Persentase Kompresi: " +
String.format("%.2f", (1 - Utils.getFileSize(outputPath) / (double)
Utils.getFileSize(inputPath)) * 100) + "%");

//              12. [OUTPUT] kedalaman pohon
            System.out.println("5. Kedalaman Pohon: " + tree.maxDepth());

//              13. [OUTPUT] banyak simpul pada pohon
            System.out.println("6. Jumlah Simpul: " + tree.leafCount());
//              14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah
```

```
ditentukan
            System.out.println("7. Gambar Tersimpan: " + outputPath);

//            15. [OUTPUT] GIF proses kompresi pada alamat yang sudah
ditentukan (bonus)
            if (generateGif) {
                System.out.println("8. GIF Tersimpan: " + gifOutputPath);
            }
            else {
                System.out.println("8. GIF tidak disimpan!");
            }


        } catch (Exception e) {
            System.out.println("Gagal!");
            e.printStackTrace();
        }

        scanner.close();
    }

    private static BinSearchResult findOptimalThreshold(int[][][] imageArray,
File inputFile, String outputPath, double targetCompression, int errorMethod)
{
        int minBlockSize = 1;
        double lowerThreshold;
        double upperThreshold;
        if (errorMethod == 1) {
            lowerThreshold = 0;
            upperThreshold = 128*128;
        }
        else if (errorMethod == 2 || errorMethod == 3) {
            lowerThreshold = 0;
            upperThreshold = 255;
        }
        else if (errorMethod == 4) {
            lowerThreshold = 0;
            upperThreshold = 8;
        }
        else {
            lowerThreshold = 0;
            upperThreshold = 1;
        }
        double threshold = (lowerThreshold + upperThreshold) / 2.0;
        double temp;
        double inputSize = inputFile.length();
        Quadtree tree = null;
        int i;
        for (i = 0; i < 100; i++) {
            if (tree != null) {
                tree = null;
                System.gc();
            }
            tree = new Quadtree(imageArray, 0, 0, imageArray[0].length,
imageArray.length, threshold, minBlockSize, errorMethod, 1);
            try {
```

```
                Utils.saveQuadtreeAsImage(tree, imageArray[0].length,
imageArray.length, outputPath);
            } catch (Exception e) {
                throw new RuntimeException(e);
            }

            double compressionRatio = 1 - Utils.getFileSize(outputPath) /
inputSize;
            if (Math.abs(compressionRatio - targetCompression) < 0.003) {
                break;
            }

            if (errorMethod == 5) {
                if (compressionRatio > targetCompression) {
                    lowerThreshold = threshold;
                } else {
                    upperThreshold = threshold;
                }
            }
            else {
                if (compressionRatio < targetCompression) {
                    lowerThreshold = threshold;
                } else {
                    upperThreshold = threshold;
                }
            }

            temp = (lowerThreshold + upperThreshold) / 2.0;
            if (Math.abs(temp - threshold) < 0.000001) {
                break;
            }
            threshold = temp;
        }
        return new BinSearchResult(threshold, tree);
    }
}
```

BAB V
IMPLEMENTASI BONUS

5.1. Target Kompresi

Dalam tugas kecil ini, penulis berhasil mengimplementasikan bonus target persentase kompresi. Bonus ini dapat diuji dengan memasukkan nilai real dalam selang (0,1] yang menyatakan target persentase kompresi. Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai threshold secara otomatis untuk mencapai target persentase kompresi yang ditentukan.

Penyesuaian threshold secara dinamis diimplementasikan dalam fungsi `findOptimalThreshold()`. Penyesuaian threshold dilakukan dengan mendeklarasikan threshold maksimum dan minimum:

- [0,16384] untuk metode perhitungan galat berdasarkan variansi
- [0,255] untuk metode perhitungan galat berdasarkan MAD dan MPD
- [0,8] untuk metode perhitungan galat berdasarkan entropi
- [0,1] untuk metode perhitungan galat berdasarkan SSIM

kemudian melakukan binary search untuk mendapatkan threshold optimal. Binary search dilakukan dengan mencoba nilai tengah dari rentang threshold yang mungkin, kemudian mengkompres gambar dan membandingkan hasilnya dengan target kompresi. Jika threshold terlalu besar, batas atas threshold yang mungkin akan diturunkan, threshold diperbarui dengan nilai tengah baru, dan gambar dikompres ulang. Begitu juga jika threshold terlalu kecil, batas bawah threshold yang mungkin akan dinaikkan. Proses ini dilakukan berulang hingga mendapatkan persentase kompresi sesuai target. Sedangkan, dalam mode ini, parameter minimum block size selalu bernilai 1.

5.2. Structural Similarity Index (SSIM)

Dalam tugas kecil ini, penulis berhasil mengimplementasikan bonus metode pengukuran galat Structural Similarity Index (SSIM). Bonus ini dapat diuji dengan memasukkan angka ‘5’ pada pilihan Metode Perhitungan Galat. Ketika mode ini diaktifkan, penentuan pembelahan pohon akan dilakukan dengan menghitung SSIM.

SSIM adalah indeks yang mengukur kemiripan dari dua gambar secara objektif. Dalam mengukur kemiripan dua gambar, SSIM mempertimbangkan tiga faktor utama:

1. Luminance (Kecerahan), yaitu perbandingan rata-rata intensitas piksel antara dua gambar
2. Contrast (Kontras), yaitu perbandingan perbedaan intensitas antara dua gambar
3. Structure (Struktur), yaitu perbandingan pola piksel antara dua gambar

Dalam tugas ini, SSIM digunakan untuk mengukur kesamaan antara gambar asli yang tercakup dalam blok (X) dengan gambar hasil kompresi yaitu gambar monoton dengan warna rata-rata gambar asli (Y).

Dalam tugas ini, SSIM dihitung menggunakan rumus

$$\mathrm{SSIM}(\mathbf{x},\mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{\left(\mu_x^2 + \mu_y^2 + C_1\right)\left(\sigma_x^2 + \sigma_y^2 + C_2\right)}.$$

Karena Y adalah gambar yang hanya memiliki satu nilai warna di seluruh pikselnya, maka $\sigma_Y = \sigma_{XY} = 0$. Selain itu, nilai warna untuk Y diambil dari nilai rata-rata warna X sehingga $\mu_Y = \mu_X$. Oleh karena itu, rumus SSIM(X,Y) dapat disederhanakan menjadi

$$SSIM(X,Y) = \frac{C_2}{\sigma_X^{\ 2} + C_2}$$

dengan $\sigma_X^{\ 2}$ adalah variansi X dan C_2 adalah konstanta yang ditentukan dengan rumus $C_2 = (K_2 L)^2$ dengan $K_2 \ll 1$ dan L adalah rentang nilai piksel, dalam hal ini $L = 255$ untuk 8-bit tiap kanal. Dalam tugas ini, dipilih $K_2 = 0.03$ sehingga $C_2 = ((0.03)(255))^2 = 58.5225$

5.3. GIF

Dalam tugas kecil ini, penulis berhasil mengimplementasikan bonus penampilan pengompresan gambar dalam bentuk GIF. Bonus ini dapat diuji dengan memasukkan alamat absolut untuk keluaran GIF. Jika mode ini diaktifkan, program akan membuat GIF yang menampilkan proses pengompresan gambar setelah pohon terbentuk.

Pembuatan GIF dilakukan dengan membuat beberapa gambar dan menggabungkannya sebagai sekuens GIF. Gambar yang dibuat adalah gambar yang diisi dengan warna dari simpul quadtree berdasarkan kedalaman yang ditentukan. Kedalaman d untuk gambar ke-i ditentukan dengan rumus $d_i = d_0 + \frac{i(d_{max} - d_0)}{4}$. Setelah sekuens gambar terbentuk, gambar-gambar tersebut digabung menjadi GIF dengan bantuan pustaka AnimatedGIFWriter yang tertera pada lampiran. Penggunaan pustaka dilakukan dengan membuat file .jar dan menaruhnya di folder lib. Kode sumber untuk implementasi GIF dapat dilihat pada bagian Utils.java

BAB VI
EKSPERIMEN DAN ANALISIS

6.1. Pengujian Program

File keluaran dapat dilihat di dalam folder test di repositori github yang tertera di lampiran.

No	Input	Screenshot

1	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\01_test.jpg	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\01_test.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok m um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukk ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosong jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 1311 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 796,83 KB 4. Persentase Kompresi: 38,40% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 137885 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\01_test.jpg 8. GIF tidak disimpan!
2	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 2 15 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\02_test.jpg	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 2 15 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\02_test.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok m um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukk ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosong jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 1306 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 697,35 KB 4. Persentase Kompresi: 46,09% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 99485 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\02_test.jpg 8. GIF tidak disimpan!
3	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 3 40 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\03_test.jpg	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 3 40 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\03_test.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok min um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukkan ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosongka jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 1312 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 857,56 KB 4. Persentase Kompresi: 33,70% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 171505 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\03_test.jpg 8. GIF tidak disimpan!

4	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 4 6 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\04_test.jpg	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 4 6 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\04_test.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok mi um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukka ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosongk jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 4605 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 583,57 KB 4. Persentase Kompresi: 54,89% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 67717 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\04_test.jpg 8. GIF tidak disimpan!
5	D:\Syafiq\4th_Semester\Stima\Tucil2\test\riza.png 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\05_test.png	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\riza.png 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\05_test.png Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok mi um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukka ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosongk jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 655 ms 2. Sebelum: 2034,34 KB 3. Sesudah: 275,38 KB 4. Persentase Kompresi: 86,46% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 37193 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\05_test.png 8. GIF tidak disimpan!
6	D:\Syafiq\4th_Semester\Stima\Tucil2\test\monalisa.jpeg 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\06_test.jpeg	Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok m um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukk ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosong jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 133 ms 2. Sebelum: 115,61 KB 3. Sesudah: 48,74 KB 4. Persentase Kompresi: 57,84% 5. Kedalaman Pohon: 8 6. Jumlah Simpul: 4361 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\06_test.jpeg 8. GIF tidak disimpan!

7	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 1 100 100 0,3 D:\Syafiq\4th_Semester\Stima\Tucil2\test\07_test.jpg	Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok m um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukk ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosong jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 16652 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 907,75 KB 4. Persentase Kompresi: 29,82% 5. Kedalaman Pohon: 13 6. Jumlah Simpul: 614637 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\07_test.jpg 8. GIF tidak disimpan!
8	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 5 0,7 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\08_test.jpg	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 5 0,7 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\08_test.jpg Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok mi um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukka ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosongk jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 1673 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 863,06 KB 4. Persentase Kompresi: 33,28% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 176785 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\08_test.jpg 8. GIF tidak disimpan!
9	D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.jpg D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.gif	Masukkan path gambar input: D:\Syafiq\4th_Semester\Stima\Tucil2\test\lion.jpg 1 225 16 0 D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.jpg D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.gif Pilih metode perhitungan error: 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) Masukkan nomor metode: Masukkan ambang batas (threshold): Masukkan ukuran blok mini um: Masukkan target persentase kompresi (0 untuk menonaktifkan mode ini): Masukkan ath gambar output: Masukkan path gambar output: Masukkan path GIF output (kosongkan jika tidak ingin menyimpan GIF): Berhasil Kompresi! 1. Waktu eksekusi: 5974 ms 2. Sebelum: 1293,54 KB 3. Sesudah: 796,83 KB 4. Persentase Kompresi: 38,40% 5. Kedalaman Pohon: 10 6. Jumlah Simpul: 137885 7. Gambar Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.jpg 8. GIF Tersimpan: D:\Syafiq\4th_Semester\Stima\Tucil2\test\09_test.gif

6.2. Analisis Kompleksitas Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree

Algoritma divide and conquer digunakan dalam membentuk Quadtree untuk mengkompresi gambar. Dalam algoritma ini, gambar direpresentasikan sebagai senarai dua dimensi yang menyimpan tiga nilai, yaitu nilai warna merah, hijau, dan biru dari tiap piksel pada

gambar. Tiap simpul pada tree menyimpan informasi bagian gambar mana yang dicakup serta satu nilai warna berdasarkan rata-rata nilai warna dari tiap piksel pada blok tersebut. Proses konstruksi quadtree dimulai dengan akar utama berupa simpul yang mencakup seluruh gambar. Kemudian, secara rekursif, jika blok yang dicakup belum memenuhi syarat untuk direpresentasikan oleh satu nilai warna tersebut, akan dibuat empat upapohon ne, nw, sw, dan se, yang masing-masing mencakup bagian kiri-atas, kanan-atas, kiri-bawah, dan kanan-bawah dari bagian gambar yang dicakup oleh simpul saat ini. Dengan kata lain, persoalan yang ditangani oleh simpul, yaitu representasi blok gambar, dibagi secara merata menjadi empat persoalan yang lebih kecil hingga persoalan tersebut dapat diselesaikan.

Proses pembuatan sebuah quadtree dilakukan dengan pemberian nilai untuk atribut width, height, isLeaf, depth, dan root secara langsung sehingga kompleksitas waktunya tidak bergantung pada ukuran data. Dalam hal ini, algoritma bekerja dengan kompleksitas algoritma $O(1)$ dalam pemberian nilai atribut-atribut tersebut.

Proses ini juga dilakukan dengan perhitungan nilai avgColor atau rata-rata nilai warna dari semua piksel dalam blok tersebut. Perhitungan nilai rata-rata dilakukan dengan fungsi computeAverageColor() yang melintasi setiap piksel pada senarai gambar, menjumlahkan masing-masing nilai merah, hijau, dan biru dari semua piksel, kemudian membaginya sebanyak jumlah piksel. Karena komputasi dilakukan dengan melintasi setiap piksel, untuk n jumlah piksel, perhitungan nilai rata-rata dilakukan dalam $O(n)$.

Selanjutnya, setiap pembuatan quadtree akan terjadi pengecekan dengan fungsi untuk menentukan keperluan pembelahan persoalan lebih kecil. Pengecekan dilakukan dengan pemanggilan fungsi shouldSplit() yang menentukan keperluan pembelahan dengan membandingkan jumlah piksel dengan minBlockSize serta menghitung galat dan membandingkannya dengan ambang batas galat. Kompleksitas waktu pembandingan tidak bergantung pada jumlah piksel, atau bekerja dalam $O(1)$. Namun, pada setiap metode perhitungan galat, dilakukan pelintasan setiap piksel, sehingga kompleksitasnya $O(n)$. Dengan demikian, pengecekan yang meliputi perhitungan galat dan pembandingan nilai galat serta jumlah piksel bekerja dalam $O(n) + O(1) = O(n)$.

Proses pembuatan pohon yang cakupannya sudah cukup kecil akan berhenti pada pengecekan, namun pohon yang cakupannya masih terlalu besar akan dibagi menjadi empat upapohon. Pembagian dilakukan dengan membelah gambar yang dicakup pada titik tengah ordinat dan titik tengah absis secara $O(1)$. Kemudian, karena persoalan dibagi menjadi 4 sama rata, tiap upa-persoalan memiliki kompleksitas waktu T(n/4).

Jadi, proses konstruksi quadtree mencakup pemberian nilai atribut dalam $O(1)$ ditambah perhitungan rata-rata dalam $O(n)$, kemudian pengecekan kondisi dalam $O(n)$ serta rekursi pembelahan persoalan menjadi empat upa-persoalan masing-masing dalam T(n/4). Oleh karena

itu, kompleksitas waktu algoritma divide and conquer dalam kompresi gambar menggunakan quadtree dapat ditulis sebagai

$$T(n) = O(1) + O(n) + O(n) + 4T(n/4) = 4T(n/4) + O(2n + 1)$$

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

Menurut Teorema Master, algoritma dengan kompleksitas waktu $T(n) = aT(n/b) + cn^d$ maka $T(n)$ adalah $O(n^d log(n))$ jika $a = b^d$. Untuk persoalan quadtree, $T(n) = 4T(n/4) + O(n)$ dapat dianggap sebagai $T(n) = 4T(n/4) + cn^1$ karena koefisien c tidak diperhatikan. Dengan demikian, didapat $a = 4$, $b = 4$, $d = 1$ sehingga memenuhi $a = b^d$. Oleh karena itu, kompleksitas waktu algoritma divide and conquer dalam kompresi gambar dengan metode quadtree adalah $T(n) = O(n.log(n))$.

6.2. Analisis Kompleksitas Algoritma Penentuan Ambang Batas Optimal untuk Target Persentase Kompresi

Penentuan ambang batas optimal untuk target persentase kompresi diimplementasikan dalam fungsi `findOptimalThreshold()`. Penentuan ambang batas optimal dilakukan dengan pertama memberi nilai 1 untuk minBlockSize dan memberi nilai batas bawah dan batas atas dari ambang batas (floor dan ceiling dari threshold) berdasarkan metode perhitungan galat yang digunakan.

- Untuk metode varians, batas bawah adalah 0 dan batas atas adalah 128^2 sesuai dengan nilai minimum dan maksimum dari varians untuk kanal warna bernilai 0-255
- Untuk metode MAD dan MPD, batas bawah adalah 0 dan batas atas adalah 255 sesuai dengan nilai minimum dan maksimum dari MAD/MPD untuk kanal warna bernilai 0-255
- Untuk metode entropi, batas bawah adalah 0 dan batas atas adalah 255 sesuai dengan nilai minimum dan maksimum dari entropi untuk kanal warna berukuran 8 bit
- Untuk metode SSIM, batas bawah adalah 0 dan batas atas adalah 1 sesuai rentang SSIM
- Kompleksitas waktu pemberian nilai ini adalah $O(1)$. Selain itu, ukuran gambar asli dihitung dalam $O(n)$ dengan n adalah jumlah piksel pada gambar dan disimpan sebelum memasuki loop.

Selanjutnya, loop dijalankan untuk mencari ambang batas yang optimal untuk mencapai target kompresi. Pencarian dilakukan dengan binary search:

- Memilih nilai tengah dari batas bawah dan batas atas
- Jika nilai tengah terlalu kecil, batas bawah dinaikkan ke nilai tengah
- Jika nilai tengah terlalu besar, batas atas diturunkan ke nilai tengah
- Mengulangi proses hingga didapat nilai sesuai target
- Pencarian ini memiliki kompleksitas waktu $O(log(m))$ dengan m rentang nilai ambang batas yang mungkin

Pada setiap iterasinya, dilakukan konstruksi quadtree dalam $O(n.log(n))$, konstruksi gambar dari quadtree dalam $O(n)$, perhitungan ukuran gambar dalam $O(n)$ dan perbandingan persentase kompresi dalam $O(1)$. Dengan demikian, kompleksitas waktu algoritma penentuan ambang batas optimal untuk target persentase kompresi adalah $O(log(m) \times (O(n.log(n)) + O(n) + O(n) + O(1)) = O(n.log(n).log(m))$

LAMPIRAN

Lampiran Checklist Spesifikasi Program

Tabel 5.1. Tabel *checklist* spesifikasi program

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

Lampiran Repository Program

Repository program dapat dibuka lewat tautan berikut:
https://github.com/iammadsfq/Tucil2_13523135

Pustaka pemroses sekuens gambar menjadi GIF didapat dari:
https://github.com/dragon66/animated-gif-writer