

Laporan Tugas Kecil 02
IF2211 STRATEGI ALGORITMA



Disusun oleh:

Rafa Abdussalam Danadyaksa (13523133)

Ardell Aghna Mahendra (13523151)

PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2025

Daftar Isi

Daftar Isi	2
BAB I DESKRIPSI MASALAH	3
BAB II TEORI SINGKAT	4
2.1. Algoritma Divide and Conquer	4
2.2. Quadtree	4
BAB III IMPLEMENTASI PROGRAM	6
3.1. Algoritma divide and conquer yang Digunakan	8
3.2. Source Code Program	9
3.3. Alur Kerja Program	22
BAB IV HASIL DAN ANALISIS	23
4.1 Hasil	23
4.2 Analisis	28
BAB V PENUTUP	30
5.1. Kesimpulan	30
5.2. Saran	30
5.3. Komentar	30
LAMPIRAN	31
Link Repository	31
Tabel Checkpoint	31
DAFTAR REFERENSI	32

BAB I

DESKRIPSI MASALAH

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB. 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. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. Untuk itu, pada tugas kecil 2 ini, kami 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 seluruh parameter yang telah disebutkan sebagai user input.

BAB II

TEORI SINGKAT

2.1. Algoritma Divide and Conquer

Divide and Conquer dulunya adalah strategi militer yang dikenal dengan nama divide ut imperes. Sekarang strategi tersebut menjadi strategi fundamental di dalam ilmu komputer dengan nama Divide and Conquer. Algoritma Divide and Conquer didefinisikan sebagai berikut:

- Divide: membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama).
- Conquer: menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).
- Combine: menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Sehingga dapat dikatakan bahwa Divide and Conquer adalah algoritma yang berprinsip memecah-mecah permasalahan yang terlalu besar menjadi beberapa bagian kecil sehingga lebih mudah untuk diselesaikan dan nantinya akan digabung kembali untuk mendapat solusi paling sempurna.

2.2. Quadtree

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

seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

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

BAB III

IMPLEMENTASI PROGRAM

Program kompresi gambar berbasis *quadtree* ini menggunakan bahasa pemrograman Java dan menggunakan algoritma *divide and conquer*. Dalam implementasi kompresi gambar berbasis Quadtree ini, digunakan beberapa metode pengukuran error untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Metode-metode tersebut diantaranya adalah Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, dan Entropy. Pengukuran error ini penting dalam algoritma *divide and conquer*, karena nilai error dibandingkan dengan threshold untuk menentukan apakah suatu blok gambar perlu dibagi lebih lanjut atau sudah cukup homogen untuk direpresentasikan dengan satu warna rata-rata. Metode-metode tersebut diimplementasikan pada program sesuai dengan rumus yang tercantum dalam spesifikasi tugas.

PSEUDOCODE

Variance

```
function Variance(image, x, y, width, height):
    avgColor = AverageColor(image, x, y, width, height)
    sumSqDiffRed = 0, sumSqDiffGreen = 0, sumSqDiffBlue = 0
    pixels = 0

    for i = x to x + width - 1:
        for j = y to y + height - 1:
            rgb = getPixel(image, i, j)
            red = getRed(rgb)
            green = getGreen(rgb)
            blue = getBlue(rgb)

            sumSqDiffRed += (red - avgColor.red)2
            sumSqDiffGreen += (green - avgColor.green)2
            sumSqDiffBlue += (blue - avgColor.blue)2
            pixels++

    redVariance = sumSqDiffRed / pixels
    greenVariance = sumSqDiffGreen / pixels
    blueVariance = sumSqDiffBlue / pixels

    return (redVariance + greenVariance + blueVariance) / 3
```

Mean Absolute Deviation (MAD)

```
function MeanAbsoluteDeviation(image, x, y, width, height):
    avgColor = AverageColor(image, x, y, width, height)
    sumAbsDiffRed = 0, sumAbsDiffGreen = 0, sumAbsDiffBlue = 0
    pixels = 0

    for i = x to x + width - 1:
        for j = y to y + height - 1:
```

```

        rgb = getPixel(image, i, j)
        red = getRed(rgb)
        green = getGreen(rgb)
        blue = getBlue(rgb)

        sumAbsDiffRed += |red - avgColor.red|
        sumAbsDiffGreen += |green - avgColor.green|
        sumAbsDiffBlue += |blue - avgColor.blue|
        pixels++

    madRed = sumAbsDiffRed / pixels
    madGreen = sumAbsDiffGreen / pixels
    madBlue = sumAbsDiffBlue / pixels

    return (madRed + madGreen + madBlue) / 3
}

Max Pixel Difference
function MaxPixelDifference(image, x, y, width, height):
    minRed = 255, maxRed = 0
    minGreen = 255, maxGreen = 0
    minBlue = 255, maxBlue = 0

    for i = x to x + width - 1:
        for j = y to y + height - 1:
            rgb = getPixel(image, i, j)
            red = getRed(rgb)
            green = getGreen(rgb)
            blue = getBlue(rgb)

            minRed = min(minRed, red)
            maxRed = max(maxRed, red)
            minGreen = min(minGreen, green)
            maxGreen = max(maxGreen, green)
            minBlue = min(minBlue, blue)
            maxBlue = max(maxBlue, blue)

    difRed = maxRed - minRed
    difGreen = maxGreen - minGreen
    difBlue = maxBlue - minBlue

    return (difRed + difGreen + difBlue) / 3

Entropy
function Entropy(image, x, y, width, height):
    redFreq = new HashMap<Integer, Integer>()
    greenFreq = new HashMap<Integer, Integer>()
    blueFreq = new HashMap<Integer, Integer>()
    pixels = 0

    for i = x to x + width - 1:
        for j = y to y + height - 1:
            rgb = getPixel(image, i, j)
            red = getRed(rgb)
            green = getGreen(rgb)
            blue = getBlue(rgb)

            redFreq[red] = redFreq.getOrDefault(red, 0) + 1
            greenFreq[green] = greenFreq.getOrDefault(green, 0) + 1
            blueFreq[blue] = blueFreq.getOrDefault(blue, 0) + 1
            pixels++

```

```
entropy = 0

for each channel in [redFreq, greenFreq, blueFreq]:
    for each count in channel.values():
        probability = count / pixels
        entropy -= (probability * log2(probability)) / 3

return entropy
```

3.1. Algoritma *divide and conquer* yang Digunakan

Langkah awal diawali dengan inisialisasi di mana seluruh gambar sebagai satu blok dengan informasi posisi (x, y), lebar (*width*), dan tinggi (*height*). Parameter yang digunakan adalah metode pengukuran error (Variance, MAD, Max Pixel Difference, atau Entropy), nilai threshold sebagai batas toleransi error dan ukuran blok minimum yang diperbolehkan. Setelah itu masuk ke tahap *Divide* atau pembagian. Dalam tahap ini, algoritma menilai apakah sebuah blok gambar perlu dibagi menjadi empat sub-blok yang lebih kecil atau tidak. Keputusan tersebut ditentukan berdasarkan hasil dari pengukuran error blok dengan metode yang dipilih, membandingkan error dengan nilai threshold yang ditentukan, dan memastikan ukuran blok setelah dibagi tidak lebih kecil dari ukuran minimum yang ditentukan. Apabila error lebih besar dari threshold dan ukuran blok masih di atas ukuran minimum, blok gambar akan dibagi menjadi empat bagian, yaitu kuadran kiri atas (*top-left*), kuadran kanan atas (*top-right*), kuadran kiri bawah (*bottom-left*), kuadran kanan bawah (*bottom-right*).

Langkah selanjutnya adalah *Conquer*. Setelah melakukan pembagian, algoritma menerapkan langkah yang sama pada masing-masing sub-blok secara rekursif. Rekursi akan berhenti ketika salah satu kondisi terpenuhi yaitu ketika error blok berada di bawah threshold atau ukuran blok saat ini sama dengan atau lebih kecil dari ukuran blok minimum atau ukuran blok setelah dibagi akan lebih kecil dari ukuran blok minimum. Lalu, ketika rekursi berhenti, algoritma akan menghitung nilai rata-rata warna (R, G, B) dari semua piksel dalam blok dan membuat node daun (leaf node) dalam struktur Quadtree yang menyimpan informasi posisi, ukuran, dan warna rata-rata blok. Selanjutnya setelah keempat sub-blok diproses, algoritma menggabungkan (*Combine*) hasil-hasil tersebut dengan membuat node internal dalam struktur Quadtree. Node internal ini memiliki empat anak (children) yang masing-masing merepresentasikan satu sub-blok. Node internal juga menyimpan informasi posisi dan ukuran blok, serta nilai rata-rata warna yang dihitung berdasarkan rata-rata tertimbang dari keempat sub-bloknya.

Setelah Quadtree selesai dibangun, algoritma menggunakan struktur ini untuk merekonstruksi gambar hasil kompresi. Proses ini dilakukan dengan cara melalui Quadtree dari *root* dan untuk setiap node daun mengisi area yang sesuai dalam gambar hasil dengan warna rata-rata yang tersimpan pada node daun tetapi tetap sesuai ukuran blok yang direpresentasikan. Dan akhirnya dihasilkan gambar yang telah dikompresi.

PSEUDOCODE

```
private QuadTreeNode buildQuadTree(int x, int y, int width, int height){
    int[] avgColor = ErrorMeasurements.AverageColor(imageProcess, x, y, width, height);
    double error = ErrorMeasurements.calculateError(errorMethod, imageProcess, x, y, width, height);

    if (error <= threshold || width <= minBlockSize || height <= minBlockSize || width / 2 <
minBlockSize || height / 2 < minBlockSize){
        return new QuadTreeNode(x, y, width, height, avgColor[0], avgColor[1], avgColor[2]);
    }

    int halfWidth = width / 2;
    int halfHeight = height / 2;
    QuadTreeNode topLeft = buildQuadTree(x, y, halfWidth, halfHeight);
    QuadTreeNode topRight = buildQuadTree(x + halfWidth, y, width - halfWidth, halfHeight);
    QuadTreeNode bottomLeft = buildQuadTree(x, y + halfHeight, halfWidth, height - halfHeight);
    QuadTreeNode bottomRight = buildQuadTree(x + halfWidth, y + halfHeight, width - halfWidth, height -
halfHeight);
    return new QuadTreeNode(x, y, width, height, topLeft, topRight, bottomLeft, bottomRight);
}
```

3.2. Source Code Program

Main.java

File ini berisi program utama yang menangani input dari pengguna, inisialisasi objek Quadtree, dan menampilkan informasi hasil kompresi.

```
import java.io.File;
import java.util.Scanner;

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

        boolean continueProgram = true;
        while(continueProgram) {
            try {

                System.out.println("\n=====");
                System.out.println("=");
                System.out.println("=                      SELAMAT DATANG                      =");
            }
        }
    }
}
```

```

System.out.println("=
System.out.println("=          PROGRAM KOMPRESI GAMBAR DENGAN QUADTREE          =");
System.out.println("=
System.out.println("=          IF2211 Strategi Algoritma          =");
System.out.println("=
System.out.println("=====\\n");

String inputImagePath = ValidImagePath(scanner);

System.out.print("\\n=====\\n");
System.out.println("Masukkan Metode perhitungan Error:");
System.out.println("1. Metode Variance");
System.out.println("2. Metode Mean Absolute Deviation (MAD)");
System.out.println("3. Metode Max Pixel Difference");
System.out.println("4. Metode Entropy");
System.out.println("5. Keluar");
System.out.print("=====\\n");
System.out.print("Pilih metode (1-5): ");

int errorMethod = scanner.nextInt();
scanner.nextLine();

if(errorMethod == 5){
    System.out.println("Terima kasih telah menggunakan program ini!");
    break;
}else if(errorMethod < 1 || errorMethod > 5){
    System.out.println("[Error] Pilihan tidak valid. Silakan pilih antara 1-5.");
    continue;
}

double threshold = getValidDoubleInput(scanner, "Masukkan ambang batas (threshold) (nilai positif): ", 0, Double.MAX_VALUE);

int minBlockSize = getValidIntInput(scanner, "Masukkan ukuran blok minimum (nilai positif): ", 1, Integer.MAX_VALUE);

String outputImagePath = ValidOutputPath(scanner);

System.out.println("\\nMemulai proses kompresi...");
long startTime = System.currentTimeMillis();

Quadtree quadtree;

try {
    quadtree = new Quadtree(inputImagePath, errorMethod, threshold, minBlockSize);
} catch (Exception e) {
    System.out.println("[Error] Gagal memproses gambar: " + e.getMessage());
    return;
}

System.out.println("Proses kompresi selesai.");
System.out.println("Menyimpan gambar hasil kompresi ke: " + outputImagePath);

try {
    quadtree.saveCompressedImage(outputImagePath);
} catch (Exception e) {
    e.printStackTrace();
}

long endTime = System.currentTimeMillis();
long executionTime = endTime - startTime;

```

```

        double persentasekompresi = (1 - (double) quadtree.getCompressedSize()/(double)
ImageProcess.getImageSize(inputImagePath)) * 100;

        System.out.println("\n=====");
        System.out.println("=                      HASIL KOMPRESI                      =");
        System.out.println("=====");
        System.out.println("|[Output] Waktu eksekusi           : " + executionTime + " ms");
        System.out.println("|[Output] Ukuran gambar sebelum      : " +
ImageProcess.getImageSize(inputImagePath) + " bytes");
        System.out.println("|[Output] Ukuran gambar setelah      : " + quadtree.getCompressedSize() + "
bytes");
        System.out.printf("|[Output] Persentase kompresi       : %.2f%%\n", persentasekompresi);
        System.out.println("|[Output] Kedalaman pohon           : " + quadtree.getMaxDepth() + " level");
        System.out.println("|[Output] Banyak simpul pada pohon   : " + quadtree.getTotalNodes() + "
simpul");
        System.out.println("=====");

        continueProgram = ContinueProgram(scanner);

        } catch (Exception e) {
            System.out.println("[Error] Terjadi kesalahan: " + e.getMessage());
        }
    }

    private static String ValidImagePath(Scanner scanner) {
        while (true) {
            System.out.println("Masukkan nama file gambar atau path absolut yang ingin dikompresi :");
            String path = scanner.nextLine().trim();

            File file = new File(path);
            if (!file.exists()) {
                File testImageFile = new File("../test/input/" + path);
                if (testImageFile.exists()) {
                    file = testImageFile;
                    path = testImageFile.getAbsolutePath();
                    System.out.println("Menggunakan file dari folder test: " + path);
                } else {
                    System.out.println("[Error] File tidak ditemukan. Pastikan file ada di folder test atau
masukkan path absolut.");
                    continue;
                }
            }

            if (!file.isFile()) {
                System.out.println("[Error] Path bukan file. Coba lagi.");
                continue;
            }

            String extension = path.substring(path.lastIndexOf('.') + 1).toLowerCase();
            if (!extension.matches("jpg|jpeg|png")) {
                System.out.println("[Error] Program hanya dapat menerima File dengan Format : jpg, jpeg,
png.");
                continue;
            }

            return path;
        }
    }
}

```

```

private static String ValidOutputPath(Scanner scanner) {
    while (true) {
        System.out.println("Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi
:");
        String path = scanner.nextLine().trim();

        String extension = path.substring(path.lastIndexOf('.') + 1).toLowerCase();
        if (!extension.matches("jpg|jpeg|png")) {
            System.out.println("[Error] Format file output tidak didukung. Format yang didukung: jpg,
jpeg, png.");
            continue;
        }

        if (!path.contains("/") && !path.contains("\\")) {
            File outputDir = new File("../test/output/");
            if (!outputDir.exists()) {
                outputDir.mkdirs();
            }
            path = "../test/output/" + path;
            System.out.println("File akan disimpan di: " + new File(path).getAbsolutePath());
        }

        File outputFile = new File(path);
        File directory = outputFile.getParentFile();
        if (directory != null && !directory.exists()) {
            System.out.println("Direktori tujuan tidak ada. Membuat direktori...");
            boolean created = directory.mkdirs();
            if (!created) {
                System.out.println("[Error] Gagal membuat direktori. Coba lagi dengan path lain.");
                continue;
            }
        }

        return path;
    }
}

private static int getValidIntInput(Scanner scanner, String prompt, int min, int max) {
    while (true) {
        System.out.print(prompt);
        String input = scanner.nextLine().trim();

        try {
            int value = Integer.parseInt(input);
            if (value < min || value > max) {
                System.out.println("[Error] Nilai harus antara " + min + " dan " + max + ". Silahkan
Coba lagi");
                continue;
            }
            return value;
        } catch (NumberFormatException e) {
            System.out.println("[Error] Input bukan angka integer yang valid. Silahkan coba lagi ");
        }
    }
}

private static double getValidDoubleInput(Scanner scanner, String prompt, double min, double max) {
    while (true) {
        System.out.print(prompt);
        String input = scanner.nextLine().trim();
    }
}

```

```

        try {
            double value = Double.parseDouble(input);
            if (value < min || value > max) {
                System.out.println("[Error] Nilai harus antara " + min + " dan " + max + ". Silahkan
coba lagi");
                continue;
            }
            return value;
        } catch (NumberFormatException e) {
            System.out.println("[Error] Input bukan angka yang valid. Silahkan coba lagi");
        }
    }
}

private static boolean ContinueProgram(Scanner scanner) {
    while (true) {
        System.out.print("\nApakah Anda ingin melanjutkan program? (y/n): ");
        String input = scanner.nextLine();
        if (input.equals("y") || input.equals("yes") || input.equals("Y") || input.equals("Yes")) {
            return true;
        } else if (input.equals("n") || input.equals("no") || input.equals("N") || input.equals("No")) {
            System.out.println("Terima kasih telah menggunakan program ini!");
            return false;
        } else {
            System.out.println("Input tidak valid. Masukkan yang valid berupa 'y' untuk ya atau 'n'
untuk tidak.");
        }
    }
}
}
}

```

Quadtree.java

File ini berisi implementasi struktur data Quadtree dan algoritma divide and conquer untuk membangun tree.

```

import java.awt.image.BufferedImage;
import java.io.File;

public class Quadtree {
    private final ImageProcess imageProcess;
    private final int errorMethod;
    private double threshold;
    private final int minBlockSize;
    private final QuadtreeNode root;
    private long compressedSize;

    public Quadtree(String imagePath, int errorMethod, double threshold, int minBlockSize) throws Exception
    {
        this.imageProcess = new ImageProcess(imagePath);
        this.errorMethod = errorMethod;
        this.threshold = threshold;
        this.minBlockSize = minBlockSize;
        this.root = buildQuadTree(0, 0, imageProcess.getWidth(), imageProcess.getHeight());
    }

    private QuadtreeNode buildQuadTree(int x, int y, int width, int height){
        int[] avgColor = ErrorMeasurements.AverageColor(imageProcess, x, y, width, height);
    }
}

```

```

        double error = ErrorMeasurements.calculateError(errorMethod, imageProcess, x, y, width, height);

        if (error <= threshold || width <= minBlockSize || height <= minBlockSize || width / 2 <
minBlockSize || height / 2 < minBlockSize){
            return new QuadtreeNode(x, y, width, height, avgColor[0], avgColor[1], avgColor[2]);
        }

        int halfWidth = width / 2;
        int halfHeight = height / 2;
        QuadtreeNode topLeft = buildQuadTree(x, y, halfWidth, halfHeight);
        QuadtreeNode topRight = buildQuadTree(x + halfWidth, y, width - halfWidth, halfHeight);
        QuadtreeNode bottomLeft = buildQuadTree(x, y + halfHeight, halfWidth, height - halfHeight);
        QuadtreeNode bottomRight = buildQuadTree(x + halfWidth, y + halfHeight, width - halfWidth, height -
halfHeight);
        return new QuadtreeNode(x, y, width, height, topLeft, topRight, bottomLeft, bottomRight);
    }

    public BufferedImage generateCompressedImage(){
        BufferedImage outputImage = new BufferedImage(imageProcess.getWidth(), imageProcess.getHeight(),
        BufferedImage.TYPE_INT_RGB
        );
        if (root != null){
            root.drawToImage(outputImage);
        }
        return outputImage;
    }

    public void saveCompressedImage(String outputPath) throws Exception{
        BufferedImage compressedImage = generateCompressedImage();
        imageProcess.saveImage(outputPath, compressedImage);
        File outputFile = new File(outputPath);
        if (outputFile.exists()){
            this.compressedSize = outputFile.length();
        } else{
            this.compressedSize = estimateCompressedSize();
        }
    }

    private long estimateCompressedSize(){
        int leafNodes = countLeafNodes();
        int internalNodes = getTotalNodes() - leafNodes;
        long leafNodesSize = leafNodes * (16 + 3);
        long internalNodesSize = internalNodes * (16 + 16);
        long overhead = 1024;
        return leafNodesSize + internalNodesSize + overhead;
    }

    public int countLeafNodes(){
        if (root == null){
            return 0;
        }
        return countLeafNodesRecursive(root);
    }

    private int countLeafNodesRecursive(QuadtreeNode node){
        if (node.isLeaf()){
            return 1;
        }
        return countLeafNodesRecursive(node.getTopLeft()) + countLeafNodesRecursive(node.getTopRight()) +
countLeafNodesRecursive(node.getBottomLeft()) + countLeafNodesRecursive(node.getBottomRight());
    }
}

```

```

public int getTotalNodes(){
    if (root == null){
        return 0;
    }
    return root.countNodes();
}

public int getMaxDepth(){
    if (root == null){
        return 0;
    }
    return root.getMaxDepth();
}

public long getCompressedSize(){
    return compressedSize;
}

public double getThreshold(){
    return threshold;
}

public void setThreshold(double threshold){
    this.threshold = threshold;
}
}

```

QuadtreeNode.java

File ini berisi implementasi node dalam struktur data Quadtree. Terdapat dua jenis node: node daun dan node internal.

```

import java.awt.image.BufferedImage;

public class QuadtreeNode {
    private final int x;
    private final int y;
    private final int width;
    private final int height;
    private final int avgRed;
    private final int avgGreen;
    private final int avgBlue;
    private final boolean isLeaf;
    private final QuadtreeNode topLeft;
    private final QuadtreeNode topRight;
    private final QuadtreeNode bottomLeft;
    private final QuadtreeNode bottomRight;

    public QuadtreeNode(int x, int y, int width, int height, int avgRed, int avgGreen, int avgBlue){
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.avgRed = avgRed;
        this.avgGreen = avgGreen;
    }
}

```

```

        this.avgBlue = avgBlue;
        this.isLeaf = true;
        this.topLeft = null;
        this.topRight = null;
        this.bottomLeft = null;
        this.bottomRight = null;
    }

    public QuadtreeNode(int x, int y, int width, int height, QuadtreeNode topLeft, QuadtreeNode topRight,
        QuadtreeNode bottomLeft, QuadtreeNode bottomRight){
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.isLeaf = false;
        this.topLeft = topLeft;
        this.topRight = topRight;
        this.bottomLeft = bottomLeft;
        this.bottomRight = bottomRight;
        long totalPixels = 0;
        long weightedSumRed = 0;
        long weightedSumGreen = 0;
        long weightedSumBlue = 0;
        QuadtreeNode[] children ={topLeft, topRight, bottomLeft, bottomRight};
        for (QuadtreeNode child : children){
            long pixels = child.getWidth() * child.getHeight();
            totalPixels += pixels;
            weightedSumRed += child.getAvgRed() * pixels;
            weightedSumGreen += child.getAvgGreen() * pixels;
            weightedSumBlue += child.getAvgBlue() * pixels;
        } if (totalPixels > 0){
            this.avgRed = (int) (weightedSumRed / totalPixels);
            this.avgGreen = (int) (weightedSumGreen / totalPixels);
            this.avgBlue = (int) (weightedSumBlue / totalPixels);
        } else{
            this.avgRed = 0;
            this.avgGreen = 0;
            this.avgBlue = 0;
        }
    }

    public void drawToImage(BufferedImage outputImage){
        if (isLeaf){
            int rgb = ImageProcess.createRGB(avgRed, avgGreen, avgBlue);
            for (int i = x; i < x + width; i++){
                for (int j = y; j < y + height; j++){
                    if (i < outputImage.getWidth() && j < outputImage.getHeight()){
                        outputImage.setRGB(i, j, rgb);
                    }
                }
            }
        } else{
            if (topLeft != null) topLeft.drawToImage(outputImage);
            if (topRight != null) topRight.drawToImage(outputImage);
            if (bottomLeft != null) bottomLeft.drawToImage(outputImage);
            if (bottomRight != null) bottomRight.drawToImage(outputImage);
        }
    }

    public int getX(){
        return x;
    }

```



```

    }
    public int getY(){
        return y;
    }
    public int getWidth(){
        return width;
    }
    public int getHeight(){
        return height;
    }
    public int getAvgRed(){
        return avgRed;
    }
    public int getAvgGreen(){
        return avgGreen;
    }
    public int getAvgBlue(){
        return avgBlue;
    }
    public boolean isLeaf(){
        return isLeaf;
    }
    public QuadtreeNode getTopLeft(){
        return topLeft;
    }
    public QuadtreeNode getTopRight(){
        return topRight;
    }
    public QuadtreeNode getBottomLeft(){
        return bottomLeft;
    }
    public QuadtreeNode getBottomRight(){
        return bottomRight;
    }

    public int countNodes(){
        if (isLeaf){
            return 1;
        } else{
            return 1 + topLeft.countNodes() + topRight.countNodes() + bottomLeft.countNodes() +
bottomRight.countNodes();
        }
    }

    public int getMaxDepth(){
        if (isLeaf){
            return 0;
        } else{
            return 1 + Math.max(Math.max(topLeft.getMaxDepth(), topRight.getMaxDepth()),
Math.max(bottomLeft.getMaxDepth(), bottomRight.getMaxDepth()));
        }
    }
}

```

ErrorMeasurements.java

File ini berisi implementasi dari metode pengukuran error yang digunakan untuk menentukan seberapa besar

perbedaan dalam satu blok gambar, yaitu Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, dan Entropy.

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

public class ErrorMeasurements {

    public static double calculateError(int method, ImageProcess image, int x, int y, int width, int height)
    {
        switch (method) {
            case 1:
                return Variance(image, x, y, width, height);
            case 2:
                return MeanAbsoluteDeviation(image, x, y, width, height);
            case 3:
                return MaxPixelDifference(image, x, y, width, height);
            case 4:
                return Entropy(image, x, y, width, height);
            default:
                throw new IllegalArgumentException("Invalid error calculation method");
        }
    }

    public static int[] AverageColor(ImageProcess image, int x, int y, int width, int height) {
        long sumRed = 0; long sumGreen = 0; long sumBlue = 0;
        int pixels = 0;

        for (int i = x; i < x + width; i++) {
            for (int j = y; j < y + height; j++) {
                int rgb = image.getPixel(i, j);
                sumRed += ImageProcess.getRed(rgb);
                sumGreen += ImageProcess.getGreen(rgb);
                sumBlue += ImageProcess.getBlue(rgb);
                pixels++;
            }
        }

        return new int[] {
            (int) (sumRed / pixels), (int) (sumGreen / pixels), (int) (sumBlue / pixels)
        };
    }

    private static double Variance(ImageProcess image, int x, int y, int width, int height){
        int[] avgColor = AverageColor(image, x, y, width, height);
        int avgRed = avgColor[0];
        int avgGreen = avgColor[1];
        int avgBlue = avgColor[2];

        double sumSqDiffRed = 0;
        double sumSqDiffGreen = 0;
        double sumSqDiffBlue = 0;
        int pixels = 0;

        for (int i = x; i < x + width && i < image.getWidth(); i++) {
            for (int j = y; j < y + height && j < image.getHeight(); j++) {
                int rgb = image.getPixel(i, j);
                int red = ImageProcess.getRed(rgb);
```

```

        int green = ImageProcess.getGreen(rgb);
        int blue = ImageProcess.getBlue(rgb);

        sumSqDiffRed += Math.pow(red - avgRed, 2);
        sumSqDiffGreen += Math.pow(green - avgGreen, 2);
        sumSqDiffBlue += Math.pow(blue - avgBlue, 2);
        pixels++;
    }
}

double redVariance = sumSqDiffRed / pixels;
double greenVariance = sumSqDiffGreen / pixels;
double blueVariance = sumSqDiffBlue / pixels;

return (redVariance + greenVariance + blueVariance) / 3;
}

private static double MeanAbsoluteDeviation(ImageProcess image, int x, int y, int width, int height){
    int[] avgColor = AverageColor(image, x, y, width, height);
    int avgRed = avgColor[0];
    int avgGreen = avgColor[1];
    int avgBlue = avgColor[2];

    double sumAbsDiffRed = 0;
    double sumAbsDiffGreen = 0;
    double sumAbsDiffBlue = 0;
    int pixels = 0;

    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            int rgb = image.getPixel(i, j);
            int red = ImageProcess.getRed(rgb);
            int green = ImageProcess.getGreen(rgb);
            int blue = ImageProcess.getBlue(rgb);

            sumAbsDiffRed += Math.abs(red - avgRed);
            sumAbsDiffGreen += Math.abs(green - avgGreen);
            sumAbsDiffBlue += Math.abs(blue - avgBlue);
            pixels++;
        }
    }

    double madRed = sumAbsDiffRed / pixels;
    double madGreen = sumAbsDiffGreen / pixels;
    double madBlue = sumAbsDiffBlue / pixels;

    return (madRed + madGreen + madBlue) / 3;
}

private static double MaxPixelDifference(ImageProcess image, int x, int y, int width, int height){
    int minRed = 255, maxRed = 0;
    int minGreen = 255, maxGreen = 0;
    int minBlue = 255, maxBlue = 0;

    for (int i = x; i < x + width && i < image.getWidth(); i++) {
        for (int j = y; j < y + height && j < image.getHeight(); j++) {
            int rgb = image.getPixel(i, j);
            int red = ImageProcess.getRed(rgb);
            int green = ImageProcess.getGreen(rgb);
            int blue = ImageProcess.getBlue(rgb);

```

```

        minRed = Math.min(minRed, red);
        maxRed = Math.max(maxRed, red);
        minGreen = Math.min(minGreen, green);
        maxGreen = Math.max(maxGreen, green);
        minBlue = Math.min(minBlue, blue);
        maxBlue = Math.max(maxBlue, blue);
    }
}

double difRed = maxRed - minRed;
double difGreen = maxGreen - minGreen;
double difBlue = maxBlue - minBlue;

return (difRed + difGreen + difBlue) / 3;
}

private static double Entropy(ImageProcess image, int x, int y, int width, int height){
    Map<Integer, Integer> redFreq = new HashMap<>();
    Map<Integer, Integer> greenFreq = new HashMap<>();
    Map<Integer, Integer> blueFreq = new HashMap<>();
    int pixels = 0;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            int rgb = image.getPixel(i, j);
            int red = ImageProcess.getRed(rgb);
            int green = ImageProcess.getGreen(rgb);
            int blue = ImageProcess.getBlue(rgb);

            redFreq.put(red, redFreq.getOrDefault(red, 0) + 1);
            greenFreq.put(green, greenFreq.getOrDefault(green, 0) + 1);
            blueFreq.put(blue, blueFreq.getOrDefault(blue, 0) + 1);
            pixels++;
        }
    }

    double entropy = 0;

    List<Map<Integer, Integer>> freqs = Arrays.asList(redFreq, greenFreq, blueFreq);
    for (Map<Integer, Integer> freq : freqs) {
        for (int count : freq.values()) {
            double probability = (double) count / pixels;
            entropy -= (probability * (Math.log(probability) / Math.log(2))) / 3;
        }
    }

    return entropy;
}
}

```

ImageProcess.java

File ini berisi implementasi untuk memproses gambar yaitu membaca dan menyimpan gambar, serta piksel gambar.

```
import java.awt.image.BufferedImage;
```

```

import java.io.File;
import java.io.IOException;
import javax.imageio.ImageIO;

public class ImageProcess {
    private final BufferedImage image;
    private int width;
    private int height;

    public ImageProcess(String imagePath) throws IOException {
        File imageFile = new File(imagePath);
        this.image = ImageIO.read(imageFile);
        this.width = image.getWidth();
        this.height = image.getHeight();
    }

    public BufferedImage getImage() {
        return image;
    }

    public int getPixel(int x, int y) {
        return image.getRGB(x, y);
    }

    public int getWidth (){
        return width;
    }

    public int getHeight (){
        return height;
    }

    public static int getRed(int rgb) {
        return (rgb >> 16) & 0xFF;
    }

    public static int getGreen(int rgb) {
        return (rgb >> 8) & 0xFF;
    }

    public static int getBlue(int rgb) {
        return rgb & 0xFF;
    }

    public static int createRGB(int r, int g, int b) {
        return ((r & 0xFF) << 16) | ((g & 0xFF) << 8) | (b & 0xFF);
    }

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

    public static void saveImage(String outputPath, BufferedImage outputImage) throws IOException {
        File outputFile = new File(outputPath);
        String extension = outputPath.substring(outputPath.lastIndexOf('.') + 1);
        ImageIO.write(outputImage, extension, outputFile);
    }
}

```

3.3. Alur Kerja Program

```
=====
=                                     =
=               SELAMAT DATANG        =
=                                     =
=      PROGRAM KOMPRESI GAMBAR DENGAN QUADTREE      =
=                                     =
=               IF2211 Strategi Algoritma           =
=                                     =
=====
```

Alur kerja program kompresi gambar kami dimulai dengan program menampilkan pesan selamat datang dan menerima input dari pengguna berupa nama file/alamat gambar yang akan dikompresi. Selanjutnya ditampilkan menu metode perhitungan error dan pengguna dapat memilih metode perhitungan error yang akan digunakan, terdapat 4 metode yaitu Variance, MAD, Max Pixel Difference, atau Entropy. Selanjutnya, program akan meminta pengguna untuk memasukkan nilai threshold sebagai batas toleransi error dan ukuran blok minimum yang akan digunakan dalam proses kompresi. Setelah itu, program meminta nama file/alamat untuk menyimpan gambar hasil kompresi.

Proses selanjutnya yaitu program memulai proses kompresi dengan membaca gambar input dan membangun struktur Quadtree menggunakan algoritma Divide and Conquer yang telah dijelaskan pada 3.1. Setiap blok gambar dievaluasi berdasarkan metode pengukuran error yang dipilih dan dibagi menjadi sub-blok jika diperlukan. Proses ini berlanjut secara rekursif hingga seluruh gambar ter-representasi dalam struktur Quadtree.

Quadtree yang telah dibangun kemudian digunakan untuk menghasilkan gambar terkompresi, di mana setiap blok homogen direpresentasikan dengan warna rata-rata dari piksel-piksel dalam blok tersebut. Gambar hasil kompresi kemudian disimpan oleh program ke nama file/alamat yang telah ditentukan oleh pengguna. Setelah proses kompresi selesai, program menampilkan informasi terkait hasil kompresi, diantaranya adalah waktu eksekusi, ukuran gambar sebelum kompresi, ukuran gambar setelah kompresi, persentase kompresi, kedalaman pohon, dan jumlah simpul pada pohon. Terakhir, program menampilkan pesan pertanyaan apakah ingin melanjutkan program (kompresi gambar lain) atau tidak. Jika user memilih ya program akan balik lagi ke alur awal, namun jika tidak akan ditampilkan pesan “Terima kasih telah menggunakan program ini!”.

BAB IV

HASIL DAN ANALISIS

4.1 Hasil

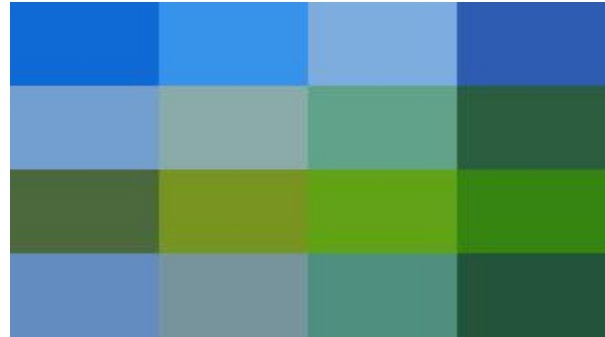
Data Uji 1 menggunakan Metode Variance dengan threshold 10 dan ukuran blok minimum 30



```
Pilih metode (1-5): 1
Masukkan ambang batas (threshold) (nilai positif): 10
Masukkan ukuran blok minimum (nilai positif): 30
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil1.jpeg
File akan disimpan di: C:\Users\rafaa\IT8\Tubes2Stima\Tucil2_13523133_13523151\src\..\test\output\Hasil1.jpeg

Memulai proses kompresi...
Proses kompresi selesai.
Menyimpan gambar hasil kompresi ke: ../test/output/Hasil1.jpeg
```

Gambar 4.1 Data uji 1



```
=====
=                               HASIL KOMPRESI                               =
=====

[[Output] waktu eksekusi           :220 ms
[[Output] Ukuran gambar sebelum    :11754 bytes
[[Output] Ukuran gambar setelah    :3523 bytes
[[Output] Persentase kompresi      : 70,03%
[[Output] Kedalaman pohon          :2 level
[[Output] Banyak simpul pada pohon :21 simpul
=====
```

Gambar 4.2 Hasil data uji 1

Data Uji 2 menggunakan Metode Mean Absolute Deviation dengan threshold 9 dan ukuran blok minimum 17



```

Masukkan nama file gambar atau path absolut yang ingin dikompresi :
GambarBunga.jpg
Menggunakan file dari folder test: /Users/ardell/Tucil2_13523133_13523151/bin/./test/input/GambarBunga.jpg

Masukkan Metode perhitungan Error:
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Difference
4. Metode Entropy
5. Keluar
=====
Pilih metode (1-5): 2
Masukkan ambang batas (threshold) (nilai positif): 9
Masukkan ukuran blok minimum (nilai positif): 17
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil2.jpg
File akan disimpan di: /Users/ardell/Tucil2_13523133_13523151/bin/./test/output/Hasil2.jpg
  
```

Gambar 4.3 Data uji 2



```

=====
HASIL KOMPRESI
=====
[Output] Waktu eksekusi           :285 ms
[Output] Ukuran gambar sebelum    :58427 bytes
[Output] Ukuran gambar setelah    :15042 bytes
[Output] Persentase kompresi      :74.26%
[Output] Kedalaman pohon          :4 level
[Output] Banyak simpul pada pohon :317 simpul
=====
  
```

Gambar 4.4 Hasil Data uji 2

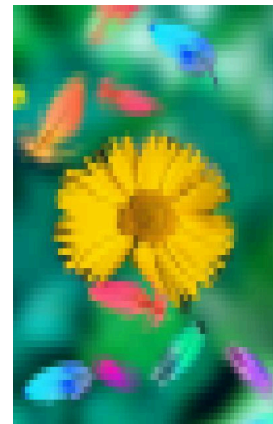
Data Uji 3 menggunakan Metode Max Pixel Difference dengan threshold 10 dan ukuran blok minimum 5



```


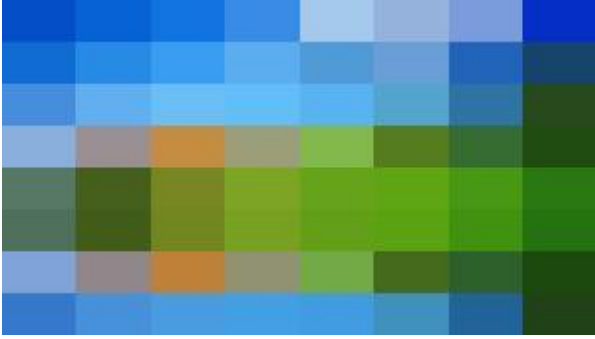


=====
Pilih metode (1-5): 3
Masukkan ambang batas (threshold) (nilai positif): 10
Masukkan ukuran blok minimum (nilai positif): 5
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil3.jpg
File akan disimpan di: C:\Users\rafaa\ITB\Tubes2Stima\Tucil2_13523133_13523151\src\..\test\output\Hasil3.jpg
File akan disimpan di: C:\Users\rafaa\ITB\Tubes2Stima\Tucil2_13523133_13523151\src\..\test\output\Hasil3.jpg

Memulai proses kompresi...
Proses kompresi selesai.
Menyimpan gambar hasil kompresi ke: ../test/output/Hasil3.jpg
  
```



```

=====
HASIL KOMPRESI
=====
[Output] Waktu eksekusi           :216 ms
[Output] Ukuran gambar sebelum    :58427 bytes
[Output] Ukuran gambar setelah    :29568 bytes
[Output] Persentase kompresi      : 49,39%
[Output] Kedalaman pohon          :6 level
[Output] Banyak simpul pada pohon :5089 simpul
=====
  
```


Gambar 4.5 Data uji 3	Gambar 4.6 Hasil data uji 3
Data Uji 4 menggunakan Metode Entropy dengan threshold 0.5 dan ukuran blok minimum 20	
 <pre data-bbox="207 678 797 924">Pilih metode (1-5): 4 Masukkan ambang batas (threshold) (nilai positif): 0.5 Masukkan ukuran blok minimum (nilai positif): 20 Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi : Hasil4.jpeg File akan disimpan di: C:\Users\rafaa\ITB\Tubes2Stima\Tucil2_13523133_13523151\src\..\test\output\Hasil4.jpeg Memulai proses kompresi... Proses kompresi selesai. Menyimpan gambar hasil kompresi ke: ../test/output/Hasil4.jpeg</pre>	 <pre data-bbox="829 678 1419 924">===== = HASIL KOMPRESI = ===== [[Output] Waktu eksekusi :203 ms [[Output] Ukuran gambar sebelum :11754 bytes [[Output] Ukuran gambar setelah :4833 bytes [[Output] Persentase kompresi : 58,88% [[Output] Kedalaman pohon :3 level [[Output] Banyak simpul pada pohon :85 simpul =====</pre>
Gambar 4.7 Data uji 4	Gambar 4.8 Hasil Data uji 4
Data Uji 5 menggunakan Metode Entropy dengan threshold 2.5 dan ukuran blok minimum 2	
	

```

Masukkan nama file gambar atau path absolut yang ingin dikompresi :
GambarNaruto.jpg
Menggunakan file dari folder test: /Users/ardell/Tucil2_13523133_13523151/bin/./test/input/GambarNaruto.jpg

=====
Masukkan Metode perhitungan Error:
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Difference
4. Metode Entropy
5. Keluar
=====
Pilih metode (1-5): 4
Masukkan ambang batas (threshold) (nilai positif): 2.5
Masukkan ukuran blok minimum (nilai positif): 2
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil5.jpg
File akan disimpan di: /Users/ardell/Tucil2_13523133_13523151/bin/./test/output/Hasil5.jpg

```

Gambar 4.9 Data uji 5

```

=====
HASIL KOMPRESI
=====
|[Output] Waktu eksekusi           :2671 ms
|[Output] Ukuran gambar sebelum    :291443 bytes
|[Output] Ukuran gambar setelah    :256062 bytes
|[Output] Persentase kompresi      :12.14%
|[Output] Kedalaman pohon          :9 level
|[Output] Banyak simpul pada pohon :257025 simpul
=====

```

Gambar 4.10 Hasil data uji 5

Data Uji 6 menggunakan Metode Mean Absolute Deviation dengan threshold 10 dan ukuran blok minimum 25



```

Masukkan nama file gambar atau path absolut yang ingin dikompresi :
GambarNaruto.jpg
Menggunakan file dari folder test: /Users/ardell/Tucil2_13523133_13523151/bin/./test/input/GambarNaruto.jpg

=====
Masukkan Metode perhitungan Error:
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Difference
4. Metode Entropy
5. Keluar
=====
Pilih metode (1-5): 2
Masukkan ambang batas (threshold) (nilai positif): 10
Masukkan ukuran blok minimum (nilai positif): 25
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil6.jpg
File akan disimpan di: /Users/ardell/Tucil2_13523133_13523151/bin/./test/output/Hasil6.jpg

```

Gambar 4.11 Data uji 6



```

=====
HASIL KOMPRESI
=====
|[Output] Waktu eksekusi           :1098 ms
|[Output] Ukuran gambar sebelum    :291443 bytes
|[Output] Ukuran gambar setelah    :122306 bytes
|[Output] Persentase kompresi      :58.03%
|[Output] Kedalaman pohon          :5 level
|[Output] Banyak simpul pada pohon :1037 simpul
=====

```

Gambar 4.12 Hasil data uji 6

Data Uji 7 menggunakan Metode Variance dengan threshold 30 dan ukuran blok minimum 8



```

Masukkan nama file gambar atau path absolut yang ingin dikompresi :
GambarPausOrca.jpg
Menggunakan file dari folder test: /Users/ardell/Tucil2_13523133_13523151/bin/./test/input/GambarPausOrca.jpg

Masukkan Metode perhitungan Error:
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Difference
4. Metode Entropy
5. Keluar

Pilih metode (1-5): 1
Masukkan ambang batas (threshold) (nilai positif): 30
Masukkan ukuran blok minimum (nilai positif): 0
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil7.jpg
File akan disimpan di: /Users/ardell/Tucil2_13523133_13523151/bin/./test/output/Hasil7.jpg
  
```

Gambar 4.13 Data uji 7



```

===== HASIL KOMPRESI =====
[Output] Waktu eksekusi           :847 ms
[Output] Ukuran gambar sebelum    :266838 bytes
[Output] Ukuran gambar setelah   :198273 bytes
[Output] Persentase kompresi      :25.70%
[Output] Kedalaman pohon         :7 level
[Output] Banyak simpul pada pohon :16345 simpul
  
```

Gambar 4.14 Hasil data uji 7

Data Uji 8 menggunakan Metode Max Pixel Difference dengan threshold 8 dan ukuran blok minimum 30



```

Masukkan nama file gambar atau path absolut yang ingin dikompresi :
GambarPausOrca.jpg
Menggunakan file dari folder test: /Users/ardell/Tucil2_13523133_13523151/bin/./test/input/GambarPausOrca.jpg

Masukkan Metode perhitungan Error:
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Difference
4. Metode Entropy
5. Keluar

Pilih metode (1-5): 3
Masukkan ambang batas (threshold) (nilai positif): 8
Masukkan ukuran blok minimum (nilai positif): 30
Masukkan nama file atau path absolut untuk menyimpan gambar hasil kompresi :
Hasil8.jpg
File akan disimpan di: /Users/ardell/Tucil2_13523133_13523151/bin/./test/output/Hasil8.jpg
  
```



```

===== HASIL KOMPRESI =====
[Output] Waktu eksekusi           :702 ms
[Output] Ukuran gambar sebelum    :266838 bytes
[Output] Ukuran gambar setelah   :85642 bytes
[Output] Persentase kompresi      :67.90%
[Output] Kedalaman pohon         :5 level
[Output] Banyak simpul pada pohon :1361 simpul
  
```

4.2 Analisis

Berdasarkan implementasi algoritma divide and conquer untuk kompresi gambar dengan metode Quadtree yang telah dilakukan, kompleksitas waktu algoritma dapat dianalisis melalui relasi rekursif $T(n) = 4T(n/4) + O(n)$, di mana n adalah jumlah piksel dalam gambar. Menggunakan Teorema Master, dengan $a = 4$, $b = 4$, dan $f(n) = O(n)$, diperoleh kompleksitas waktu $O(n \log n)$. Hasil ini menunjukkan peningkatan efisiensi dibandingkan algoritma pemrosesan gambar naif yang biasanya memiliki kompleksitas $O(n^2)$. Sementara itu, kompleksitas ruang algoritma ditentukan oleh jumlah node dalam struktur Quadtree yang dibangun. Dalam kasus terburuk, jika setiap piksel membentuk node daun tersendiri, kompleksitas ruangnya adalah $O(n)$.

Dari hasil pengujian, terlihat bahwa waktu eksekusi berkorelasi dengan ukuran gambar, nilai threshold, dan ukuran blok minimum. Gambar berukuran besar dengan threshold rendah dan ukuran blok minimum kecil membutuhkan waktu pemrosesan yang lebih lama. Contohnya pengujian pada data uji 5 (gambar naruto) dengan threshold 2.5 dan ukuran blok minimum 2 menghasilkan waktu eksekusi 2,671 ms, sedangkan pada data uji 6 dengan threshold 10 dan ukuran blok minimum 25 hanya membutuhkan 1,098 ms. Ini menunjukkan bahwa parameter kompresi mempengaruhi performa algoritma.

Metode pengukuran error yang berbeda juga memberikan hasil kompresi yang bervariasi. Metode Variance pada data uji 1 dan data uji 7 memberikan persentase kompresi yang cukup baik yaitu 70.03% dan 25.70%. Metode Mean Absolute Deviation (MAD) pada data uji 2 dan data uji 6 menghasilkan kompresi tinggi, yaitu 74.26% dan 58.03%. Metode Max Pixel Difference pada data uji 3 dan data uji 8 menghasilkan kompresi moderat, yaitu 49.39% dan 67,90%. Metode Entropy pada data uji 4 dan 5 memberikan hasil yang sangat bergantung pada kompleksitas gambar, dengan kompresi antara 58.88% hingga hanya 12.14% pada gambar yang lebih detail.

Nilai threshold dan ukuran blok minimum juga memiliki pengaruh terhadap hasil kompresi. Threshold yang lebih tinggi menghasilkan persentase kompresi lebih tinggi dengan kedalaman pohon yang lebih kecil, seperti pada data uji 1 threshold 10 dengan kedalaman pohon 2 level dan 21 simpul. Kebalikannya, jika threshold lebih rendah seperti pada data uji 5 yaitu threshold 2.5, hasil yang didapat yaitu kedalaman pohon 9 level dengan 257,025 simpul. Ukuran blok minimum juga sangat berpengaruh, di mana nilai yang lebih

kecil memungkinkan algoritma menangkap lebih banyak detail tetapi dengan biaya kompresi yang lebih rendah.

BAB V

PENUTUP

5.1. Kesimpulan

Pada Tugas Kecil 2 mata kuliah IF2211 Strategi Algoritma ini, kami berhasil mengembangkan program kompresi gambar dengan quadtree menggunakan algoritma *divide and conquer*. Implementasi kompresi gambar dengan metode Quadtree ini memberikan fleksibilitas yang baik dalam menyeimbangkan ukuran file dan kualitas gambar melalui pengaturan parameter. Meskipun tidak seefisien algoritma kompresi modern seperti JPEG untuk fotografi, metode ini tetap menjadi pilihan untuk aplikasi tertentu, seperti gambar dengan area homogen yang luas atau yang membutuhkan kompresi adaptif berdasarkan kompleksitas area.

5.2. Saran

Untuk pengembangan selanjutnya, sebaiknya dapat dicoba untuk melakukan implementasi beberapa bonus seperti implementasi Structural Similarity Index (SSIM), implementasi persentase kompresi sebagai parameter tambahan, dan *output berupa GIF Visualisasi*.

5.3. Komentar

Akhirnya tugas kecil 2 ini selesai juga. Bersiap untuk tugas kecil berikutnya dan tugas besar.

LAMPIRAN

Link Repository

Link Repository Tugas Kecil 2 : https://github.com/RafaAbdussalam/Tucil2_13523133_13523151.git

Tabel Checkpoint

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

DAFTAR REFERENSI

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

<https://andikafisma.wordpress.com/algoritma-divide-and-conquer/>