

Laporan Tugas Kecil 2
IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree

Disusun oleh:

Rendi Adinata
10123083



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

BAB I

DESKRIPSI MASALAH

Apa itu 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.

Quadtree untuk Kompresi Gambar

Pada Tugas Kecil 2 kali ini, akan digunakan struktur data Quadtree dan algoritma *divide and conquer* untuk melakukan kompresi gambar dengan bermacam-macam metode/kalkulator galat dengan parameter ambang batas dan ukuran minimum blok.

BAB 2

ALGORITMA *Divide and Conquer*

Alur Program

Masukan

Program menerima alamat gambar, metode kalkulator galat, ambang batas, ukuran blok minimum dan alamat keluaran gambar dan GIF (opsional) . Pada tahap ini gambar dianggap sebagai satu blok utuh.

Langkah-langkah Algoritma

- **Proses perhitungan galat:**
Dihitung nilai galat dari blok tersebut. Jika galat melebihi ambang batas (threshold), maka blok tersebut dibagi menjadi empat sub-blok.
- **Rekursi:**
Langkah penghitungan galat dan pembagian ini diulang secara rekursif pada setiap sub-blok hingga memenuhi kriteria penghentian, yaitu:
galat blok sudah di bawah threshold, atau ukuran blok telah mencapai ukuran minimum .
- **Pembentukan gambar hasil:**
Setelah seluruh blok mencapai kondisi penghentian, setiap blok direpresentasikan sebagai warna rata-rata yang kemudian digabungkan untuk membentuk gambar terkompresi.

Keluaran

Pada akhir proses, output yang dihasilkan dapat mencakup informasi berikut:

- **Waktu Eksekusi:** Total waktu yang diperlukan untuk proses kompresi.
- **Ukuran Gambar:** Ukuran file dari gambar asli dan gambar terkompresi.
- **Persentase Kompresi:** Persentase pengurangan ukuran file gambar.
- **Kedalaman Pohon:** Kedalaman maksimum dari struktur QuadTree, yang menggambarkan seberapa dalam proses pembagian dilakukan.

- **Jumlah Simpul:** Total jumlah node (simpul) yang dihasilkan dalam QuadTree.

Algoritma *Divide and Conquer*

Proses *Divide*

Pada tahap ini, gambar dibagi secara rekursif menjadi empat kuadran atau blok. Setiap blok yang dihasilkan akan diproses sebagai unit-unit yang terpisah. Selanjutnya, untuk setiap blok yang telah dibagi, dihitung nilai galat menggunakan metode yang telah ditentukan (variansi, Mean Absolute Deviation/MAD, Max Pixel Difference dan Entropy). Apabila nilai galat pada blok tersebut melebihi ambang batas (threshold) yang telah ditetapkan, blok tersebut akan dibagi lagi menjadi empat sub-blok. Proses rekursif ini akan melakukan pembagian gambar berulang kali hingga diperoleh blok-blok dengan tingkat galat yang dapat diterima atau hingga ukuran blok mencapai batas minimum yang telah ditentukan.

Proses *Conquer*

Setelah proses pembagian selesai, setiap blok atau kuadran yang tersisa dianggap sebagai "daun" dalam struktur data pohon (QuadTree). Pada tahap ini, blok tersebut tidak dipisah lagi melainkan direpresentasikan dengan nilai warna rata-rata dari semua piksel di dalam blok tersebut. Warna rata-rata ini akan mewakili blok sebagai hasil kompresi pada bagian gambar tersebut.

Combine

Pada tahap akhir, seluruh blok yang tidak terpecah (daun pohon) digabungkan kembali untuk membentuk gambar terkompresi. Penggabungan dilakukan dengan cara menggambar kembali setiap blok daun pada posisi yang sesuai dengan nilai warna rata-ratanya, sehingga terbentuk gambar kompresi yang menyerupai gambar asli namun dengan jumlah informasi yang telah direduksi.

BAB 3

SOURCE CODE PROGRAM

VarianceCalculator

```
class VarianceCalculator implements ErrorCalculator {

    @Override

    public double calculateDetail(int[] rHist, int[] gHist, int[] bHist, int
totalPixels) {

        double rVar = calculateVariance(rHist, totalPixels);

        double gVar = calculateVariance(gHist, totalPixels);

        double bVar = calculateVariance(bHist, totalPixels);

        return 0.2989 * rVar + 0.5870 * gVar + 0.1140 * bVar;

    }

    private double calculateVariance(int[] hist, int total) {

        if (total == 0) return 0.0;

        double sum = 0.0, sumSquares = 0.0;

        for (int i = 0; i < hist.length; i++) {

            sum += i * hist[i];

            sumSquares += (i * i) * hist[i];

        }

        double mean = sum / total;

        return (sumSquares / total) - (mean * mean);

    }

}
```

MadCalculator

```
class MADCalculator implements ErrorCalculator {
```

```

@Override

    public double calculateDetail(int[] rHist, int[] gHist, int[] bHist, int
totalPixels) {

        double rMad = calculateMAD(rHist, totalPixels);

        double gMad = calculateMAD(gHist, totalPixels);

        double bMad = calculateMAD(bHist, totalPixels);

        return 0.2989 * rMad + 0.5870 * gMad + 0.1140 * bMad;

    }

private double calculateMAD(int[] hist, int total) {

    if (total == 0) return 0.0;

    double sum = 0.0;

    for (int i = 0; i < hist.length; i++)

        sum += i * hist[i];

    double mean = sum / total;

    double mad = 0.0;

    for (int i = 0; i < hist.length; i++)

        mad += Math.abs(i - mean) * hist[i];

    return mad / total;

}

}

```

MaxDiffCalculator

```

class MaxDiffCalculator implements ErrorCalculator {

    @Override

    public double calculateDetail(int[] rHist, int[] gHist, int[] bHist, int
totalPixels) {

```

```

        int rMaxDiff = calculateMaxDiff(rHist);
        int gMaxDiff = calculateMaxDiff(gHist);
        int bMaxDiff = calculateMaxDiff(bHist);

        return Math.max(rMaxDiff, Math.max(gMaxDiff, bMaxDiff));
    }

private int calculateMaxDiff(int[] hist) {
    int min = 255, max = 0;

    for (int i = 0; i < hist.length; i++) {
        if (hist[i] > 0) {
            if (i < min) min = i;
            if (i > max) max = i;
        }
    }

    return max - min;
}

}

```

EntropyCalculator

```

class EntropyCalculator implements ErrorCalculator {

    @Override

    public double calculateDetail(int[] rHist, int[] gHist, int[] bHist, int
totalPixels) {

        double rEntropy = calculateEntropy(rHist, totalPixels);
        double gEntropy = calculateEntropy(gHist, totalPixels);
        double bEntropy = calculateEntropy(bHist, totalPixels);

        return 0.2989 * rEntropy + 0.5870 * gEntropy + 0.1140 * bEntropy;
    }
}

```

```

}

private double calculateEntropy(int[] hist, int total) {

    if (total == 0) return 0.0;

    double entropy = 0.0;

    for (int i = 0; i < hist.length; i++) {

        if (hist[i] > 0) {

            double p = hist[i] / (double) total;

            entropy -= p * (Math.log(p) / Math.log(2));

        }

    }

    return entropy;

}

}

```

Kelas Quadrant

Kelas ini bertujuan untuk menghitung galat setiap blok dan menentukan apakah blok tersebut harus dipecah lebih lanjut atau tidak. Pada konstruktornya, blok dianalisis, dan apabila galat melebihi *threshold*, dilakukan proses *split* (pembagian blok) menjadi empat kuadran baru.

```

class Quadrant {

    private final Rectangle bbox;
    private final int depth;
    private Quadrant[] children;
    private boolean isLeaf;
    private final Color averageColor;
    private final double detail;

    public Quadrant(BufferedImage image, Rectangle bbox, int depth,
ErrorCalculator errorCalculator, double threshold, int minBlockSize, int
}

```

```

maxDepth) {
    this.bbox = bbox;
    this.depth = depth;

    int[] rHist = new int[256];
    int[] gHist = new int[256];
    int[] bHist = new int[256];
    int rSum = 0, gSum = 0, bSum = 0, totalPixels = 0;

    for (int y = bbox.y; y < bbox.y + bbox.height; y++) {
        for (int x = bbox.x; x < bbox.x + bbox.width; x++) {
            int pixel = image.getRGB(x, y);
            int r = (pixel >> 16) & 0xFF;
            int g = (pixel >> 8) & 0xFF;
            int b = pixel & 0xFF;

            rHist[r]++;
            gHist[g]++;
            bHist[b]++;
            rSum += r;
            gSum += g;
            bSum += b;
            totalPixels++;
        }
    }

    this.averageColor = new Color(rSum / totalPixels, gSum / totalPixels,
bSum / totalPixels);
    this.detail = errorCalculator.calculateDetail(rHist, gHist, bHist,
totalPixels);

    boolean canSplit = (bbox.width > minBlockSize && bbox.height >
minBlockSize) && (depth < maxDepth);
    if (detail <= threshold || !canSplit) {
        this.isLeaf = true;
        this.children = null;
    } else {
        this.isLeaf = false;
        split(image, errorCalculator, threshold, minBlockSize, maxDepth);
    }
}

```

```

private void split(BufferedImage image, ErrorCalculator errorCalculator,
double threshold, int minBlockSize, int maxDepth) {
    int halfWidth = bbox.width / 2;
    int halfHeight = bbox.height / 2;

    children = new Quadrant[]{
        new Quadrant(image, new Rectangle(bbox.x, bbox.y, halfWidth,
halfHeight), depth + 1, errorCalculator, threshold, minBlockSize, maxDepth),
        new Quadrant(image, new Rectangle(bbox.x + halfWidth, bbox.y,
bbox.width - halfWidth, halfHeight), depth + 1, errorCalculator, threshold,
minBlockSize, maxDepth),
        new Quadrant(image, new Rectangle(bbox.x, bbox.y + halfHeight,
halfWidth, bbox.height - halfHeight), depth + 1, errorCalculator, threshold,
minBlockSize, maxDepth),
        new Quadrant(image, new Rectangle(bbox.x + halfWidth, bbox.y +
halfHeight, bbox.width - halfWidth, bbox.height - halfHeight), depth + 1,
errorCalculator, threshold, minBlockSize, maxDepth)
    };
}

public boolean isLeaf() { return isLeaf; }
public Quadrant[] getChildren() { return children; }
public Rectangle getBbox() { return bbox; }
public Color getAverageColor() { return averageColor; }
public int getDepth() { return depth; }
}

```

Kelas QuadTree

Kelas ini menggunakan struktur rekursif untuk membangun pohon quadtree berdasarkan pembagian yang dilakukan oleh kelas Quadrant. QuadTree memastikan seluruh blok dalam gambar dianalisis secara menyeluruh, dan menyimpan informasi mengenai kedalaman pohon serta jumlah simpul (node).

```

class QuadTree {
    private final Quadrant root;
    private int maxDepth;
    private int nodeCount;

    public QuadTree(BufferedImage image, ErrorCalculator errorCalculator,
double threshold, int minBlockSize) {

```

```

        int maxDimension = Math.max(image.getWidth(), image.getHeight());
        int calculatedMaxDepth = 0;
        while (maxDimension > minBlockSize) {
            maxDimension /= 2;
            calculatedMaxDepth++;
        }

        this.root = new Quadrant(image, new Rectangle(0, 0, image.getWidth(),
image.getHeight(), 0, errorCalculator, threshold, minBlockSize,
calculatedMaxDepth);
        this.maxDepth = 0;
        this.nodeCount = 0;
        buildTree(root);
    }

    private void buildTree(Quadrant node) {
        nodeCount++;
        if (node.isLeaf()) {
            if (node.getDepth() > maxDepth)
                maxDepth = node.getDepth();
            return;
        }
        for (Quadrant child : node.getChildren())
            buildTree(child);
    }

    public BufferedImage createCompressedImage() {
        return createCompressedImage(Integer.MAX_VALUE);
    }

    public BufferedImage createCompressedImage(int maxFrameDepth) {
        BufferedImage img = new BufferedImage(root.getBbox().width,
root.getBbox().height, BufferedImage.TYPE_INT_RGB);
        Graphics2D g = img.createGraphics();
        g.setColor(Color.BLACK);
        g.fillRect(0, 0, img.getWidth(), img.getHeight());

        List<Quadrant> leaves = new ArrayList<>();
        getLeaves(root, leaves, maxFrameDepth);
        for (Quadrant leaf : leaves) {
            Rectangle rect = leaf.getBbox();
            g.setColor(leaf.getAverageColor());

```

```

        g.fillRect(rect.x, rect.y, rect.width, rect.height);
    }
    g.dispose();
    return img;
}

private void getLeaves(Quadrant node, List<Quadrant> leaves, int maxFrameDepth) {
    if (node.isLeaf() || node.getDepth() == maxFrameDepth) {
        leaves.add(node);
    } else {
        for (Quadrant child : node.getChildren())
            getLeaves(child, leaves, maxFrameDepth);
    }
}

public int getMaxDepth() { return maxDepth; }
public int getNodeCount() { return nodeCount; }
}

```

GifSequenceWriter

Implementasi pemuatan GIF dilakukan dengan mengumpulkan *frame-by-frame* proses kompresi Quadtree menggunakan algoritma *multi-depth capture*, yakni pada setiap tahap pembagian blok divisualisasikan sebagai layer terpisah. Program menghasilkan serangkaian gambar pada kedalaman berbeda (dari *root* hingga *leaf node*) dengan memanggil metode `createCompressedImage(depth)` secara iteratif, lalu menggabungkannya menjadi animasi GIF menggunakan kelas `GifSequenceWriter` dari library `javax.imageio`. Setiap frame disimpan dengan delay 500 ms dan di-loop 4 kali di akhir untuk memberikan jeda visual dan metadata GIF diatur untuk mengontrol durasi tayang dan transparansi. Frame-frame tersebut dikodekan sebagai `BufferedImage` bertipe `TYPE_INT_RGB` untuk memastikan konsistensi warna, lalu ditulis ke dalam file output via `ImageOutputStream`, menghasilkan animasi yang merekam hierarki pembagian Quadtree secara bertahap.

```

class GifSequenceWriter {
    protected ImageWriter writer;
    protected ImageWriteParam params;
    protected IIOMetadata metadata;

    public GifSequenceWriter(ImageOutputStream out, int imageType, int delay,
                           boolean loop) throws IOException {

```

```

writer = ImageIO.getImageWritersBySuffix("gif").next();
params = writer.getDefaultWriteParam();

ImageTypeSpecifier imageTypeSpecifier =
ImageTypeSpecifier.createFromBufferedImageType(imageType);
metadata = writer.getDefaultImageMetadata(imageTypeSpecifier, params);

configureMetadata(delay, loop);

writer.setOutput(out);
writer.prepareWriteSequence(null);
}

private void configureMetadata(int delay, boolean loop) throws
IIoInvalidTreeException {
    String metaFormat = metadata.getNativeMetadataFormatName();
    IIOMetadataNode root = (IIOMetadataNode)
metadata.getAsTree(metaFormat);

    // Set waktu delay
    IIOMetadataNode gce = getNode(root, "GraphicControlExtension");
    gce.setAttribute("delayTime", String.valueOf(delay / 10));
    gce.setAttribute("disposalMethod", "none");

    // Set proses looping
    if (loop) {
        IIOMetadataNode appExtensions = getNode(root,
"ApplicationExtensions");
        IIOMetadataNode appNode = new
IIOMetadataNode("ApplicationExtension");
        appNode.setAttribute("applicationID", "NETSCAPE");
        appNode.setAttribute("authenticationCode", "2.0");
        appNode.setUserObject(new byte[]{0x1, (byte) (0), (byte) (0)});
        appExtensions.appendChild(appNode);
    }

    metadata.setFromTree(metaFormat, root);
}

private IIOMetadataNode getNode(IIOMetadataNode root, String nodeName) {
    for (int i = 0; i < root.getLength(); i++) {
        if (root.item(i).getNodeName().equalsIgnoreCase(nodeName)) {

```

```

        return (IIOMetadataNode) root.item(i);
    }
}

IIOMetadataNode node = new IIOMetadataNode(nodeName);
root.appendChild(node);
return node;
}

public void writeFrame(BufferedImage img) throws IOException {
    writer.writeToSequence(new IIIOImage(img, null, metadata), params);
}

public void close() throws IOException {
    writer.endWriteSequence();
}
}

```

ImageCompression

Bagian kode ini merupakan bagian utama yang bertujuan untuk menggambar kembali seluruh daun dari struktur QuadTree. Setiap daun yang mewakili blok gambar dengan warna rata-rata, digambarkan ulang pada posisi yang sesuai sehingga menghasilkan gambar terkompreksi.

```

public class ImageCompression {

    public static void main(String[] args) {

        Scanner scanner = new Scanner(System.in);

        try {

            System.out.print("Masukkan alamat gambar: ");

            String imagePath = scanner.nextLine().trim();

            File imageFile = new File(imagePath);

            if (!imageFile.exists() || !imageFile.isFile()) {

```

```
        System.err.println("Error: Alamat gambar tidak valid atau file  
tidak ditemukan.");  
  
        return;  
  
    }  
  
  
    // Coba membaca gambar  
  
    BufferedImage image = ImageIO.read(imageFile);  
  
    if (image == null) {  
  
        System.err.println("Error: File yang diberikan bukan file  
gambar yang dapat dikenali.");  
  
        return;  
  
    }  
  
  
    System.out.print("Pilih kalkulator galat (1-4): ");  
  
    int method;  
  
    try {  
  
        method = Integer.parseInt(scanner.nextLine().trim());  
  
        if (method < 1 || method > 4) {  
  
            System.err.println("Error: Pilihan kalkulator galat harus  
antara 1 dan 4.");  
  
            return;  
  
        }  
  
    } catch (NumberFormatException ex) {  
  
        System.err.println("Error: Input kalkulator galat harus berupa  
angka.");  
  
        return;  
    }
```

```
}

System.out.print("Masukkan ambang batas: ");

double threshold;

try {

    threshold = Double.parseDouble(scanner.nextLine().trim());

} catch (NumberFormatException ex) {

    System.err.println("Error: Ambang batas harus berupa angka.");

    return;

}

System.out.print("Masukkan ukuran blok minimum: ");

int minBlockSize;

try {

    minBlockSize = Integer.parseInt(scanner.nextLine().trim());

    if (minBlockSize <= 0) {

        System.err.println("Error: Ukuran blok minimum harus lebih besar dari 0.");

        return;

    }

} catch (NumberFormatException ex) {

    System.err.println("Error: Ukuran blok minimum harus berupa angka.");

    return;

}
```

```
System.out.print("Masukkan alamat output gambar (.jpg): ");

String outputPath = scanner.nextLine().trim();

if (outputPath.isEmpty()) {

    System.err.println("Error: Alamat output gambar tidak boleh kosong.");

    return;
}

outputPath = replaceExtension(outputPath, "jpg");

System.out.print("Masukkan alamat output GIF (opsional): ");

String gifOutputPath = scanner.nextLine().trim();

if (!gifOutputPath.isEmpty()) {

    gifOutputPath = replaceExtension(gifOutputPath, "gif");
}

// Memilih kalkulator galat

ErrorCalculator errorCalculator;

switch (method) {

    case 1:

        errorCalculator = new VarianceCalculator();

        break;

    case 2:

        errorCalculator = new MADCalculator();

        break;
}
```

```
        case 3:

            errorCalculator = new MaxDiffCalculator();

            break;

        case 4:

            errorCalculator = new EntropyCalculator();

            break;

        default:

            System.err.println("Error: Input kalkulator galat tidak
valid.");
    }

    // Proses kompresi

    long startTime = System.currentTimeMillis();

    QuadTree quadTree = new QuadTree(image, errorCalculator,
threshold, minBlockSize);

    BufferedImage compressedImage = quadTree.createCompressedImage();

    long endTime = System.currentTimeMillis();

    if (!ImageIO.write(compressedImage, "jpg", new File(outputPath)))
    {

        System.err.println("Error: Gagal menyimpan gambar ke " +
outputPath);

        return;
    }
}
```

```

long originalSize = imageFile.length();

long compressedSize = new File(outputPath).length();

System.out.println("Execution time: " + (endTime - startTime) + " ms");

System.out.println("Original size: " + originalSize + " bytes");

System.out.println("Compressed size: " + compressedSize + " bytes");

System.out.printf("Compression percentage: %.2f%%\n", (1.0 -
(double) compressedSize / originalSize) * 100);

System.out.println("Tree depth: " + quadTree.getMaxDepth());

System.out.println("Node count: " + quadTree.getNodeCount());

if (!gifOutputPath.isEmpty()) {

    List<BufferedImage> frames = new ArrayList<>();

    for (int depth = 0; depth <= quadTree.getMaxDepth(); depth++)

    {
        frames.add(quadTree.createCompressedImage(depth));

    }

    BufferedImage finalFrame =
quadTree.createCompressedImage(quadTree.getMaxDepth());

    for (int i = 0; i < 4; i++) {

        frames.add(finalFrame);
    }
}

```

```

    }

    try (ImageOutputStream output =
ImageIO.createImageOutputStream(new File(gifOutputPath))) {

        GifSequenceWriter writer = new GifSequenceWriter(output,
BufferedImage.TYPE_INT_RGB, 500, true);

        for (BufferedImage frame : frames) {

            writer.writeFrame(frame);

        }

        writer.close();

    } catch (IOException e) {

        System.err.println("Error saat menulis GIF: " +
e.getMessage());

    }

}

} catch (IOException e) {

    System.err.println("Terjadi kesalahan I/O: " + e.getMessage());

} finally {

    scanner.close();

}

}

private static String replaceExtension(String path, String newExt) {

    if (path == null || path.isEmpty()) return path;

    int lastDot = path.lastIndexOf('.');
}

```

```
if (lastDot != -1) {  
  
    path = path.substring(0, lastDot);  
  
}  
  
return path + "." + newExt;  
  
}  
}
```

BAB 4

INPUT DAN OUTPUT PROGRAM

Test case 1:

Metode : Variance

Ambang batas : 50

Ukuran blok minimum: 4

Masukan	Keluaran
 <i>Di darat, rasanya seperti berdiri di bawah panggangan 400 derajat.</i>	<p>Execution time: 842 ms Original size: 1811321 bytes Compressed size: 115384 bytes Compression percentage: 93.63% Tree depth: 9 Node count: 15961</p> 

Test case 2:

Metode : Variance

Ambang batas : 9

Ukuran blok minimum: 2

Masukan	Keluaran
	<p>Execution time: 844 ms Original size: 992930 bytes Compressed size: 83178 bytes Compression percentage: 91.62% Tree depth: 9 Node count: 63489</p>  <p>link GIF: https://jmp.sh/wKvQcxG</p>

Test case 3:

Metode : Mean Absolute Deviation (MAD)

Ambang batas : 20

Ukuran blok minimum: 4

Masukan	Keluaran
---------	----------



Execution time: 577 ms
Original size: 1395786 bytes
Compressed size: 115700 bytes
Compression percentage: 91.71%
Tree depth: 8
Node count: 26849



Test case 4:

Metode : Mean Absolute Deviation (MAD)

Ambang batas : 9

Ukuran blok minimum: 2

Masukan	Keluaran
---------	----------



Execution time: 792 ms
Original size: 1087198 bytes
Compressed size: 102978 bytes
Compression percentage: 90.53%
Tree depth: 9
Node count: 33633



Link GIF:
<https://jmp.sh/DqP0jpnH>

Test case 5:

Metode : Max Pixel Difference

Ambang batas : 9

Ukuran blok minimum: 2

Masukan	Keluaran
Three young men standing in front of a glass building with a golden statue above the entrance. They are dressed in casual attire: one in a dark blue shirt, one in a grey jacket over a blue shirt, and one in a dark blue polo shirt.	<p>Execution time: 580 ms Original size: 141652 bytes Compressed size: 110888 bytes Compression percentage: 21.72% Tree depth: 8 Node count: 35341</p> A compressed version of the photograph of three young men, showing noticeable blocky artifacts and loss of detail. The background glass wall and the golden statue are less distinct.

Test case 6:

Metode : Max Pixel Difference

Ambang batas : 4

Ukuran blok minimum: 2

Masukan	Keluaran
	<p>Execution time: 1295 ms Original size: 1536681 bytes Compressed size: 109036 bytes Compression percentage: 92.90% Tree depth: 9 Node count: 278221</p>  <p>Link GIF: https://jmp.sh/w6htMJNu</p>

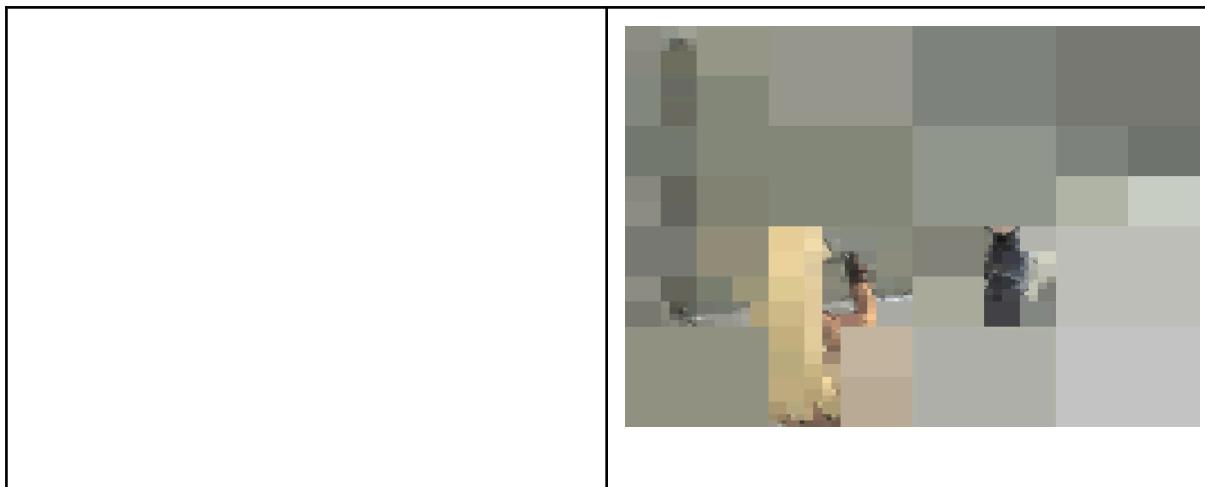
Test case 7:

Metode : Entropy

Ambang batas : 6

Ukuran blok minimum: 2

Masukan	Keluaran
	<p>Execution time: 392 ms Original size: 174066 bytes Compressed size: 47744 bytes Compression percentage: 72.57% Tree depth: 8 Node count: 745</p>



Test case 8:

Metode : Entropy

Ambang batas : 3

Ukuran blok minimum: 2

Masukan	Keluaran
 <p>Jika kau tak keberatan dengan pria sepertiku,</p>	<p>Execution time: 1033 ms Original size: 1478117 bytes Compressed size: 127931 bytes Compression percentage: 91.35% Tree depth: 9 Node count: 91497</p>  <p>Jika kau tak keberatan dengan pria sepertiku,</p> <p>Link GIF: https://jmp.sh/Njz5qqBO</p>

Bab 5

Analisis dan Pembahasan

Berdasarkan algoritma divide-and-conquer di atas, diperoleh analisis kompleksitas waktu dan ruang sebagai berikut:

1. Kompleksitas Waktu

Algoritma Quadtree menggunakan pendekatan *divide-and-conquer* dengan langkah:

- **Divide**: Membagi gambar menjadi 4 kuadran ($O(1)$).
- **Conquer**: Memproses setiap kuadran secara rekursif.
- **Combine**: Menggabungkan hasil kompresi dari kuadran.

Kasus Terburuk (Worst Case):

Jika gambar sangat tidak homogen, semua kuadran akan terus dipecah hingga mencapai ukuran blok minimum. Ini berarti setiap level rekursi membagi gambar menjadi 4 bagian. Kedalaman rekursi maksimum: $\log_2(\max(N,M))$, dengan NxM adalah resolusi gambar

$$\text{Kompleksitas: } O(4^{\log_2(\max(N,M))}) = O(\max(N,M)^2)$$

Ini setara dengan kompleksitas $O(N^2)$ untuk gambar persegi $N \times N$

Kasus Terbaik:

Jika gambar sangat homogen, tidak ada pembagian lebih lanjut.

Kompleksitas: $O(1)$

Kasus Rata-Rata :

Kompleksitas: $O(N \log N)$.

2. Kompleksitas Ruang

Kedalaman rekursi maksimum: $\log_2(\max(N,M))$

Penyimpanan node: Setiap node menyimpan data (posisi, ukuran, warna).

Kompleksitas ruang: $O(\text{jumlah node})$ dengan kasus terburuk $O(N^2)$

3. Analisis Pengaruh Parameter terhadap Kualitas Kompresi

Berdasarkan beberapa kasus yang terlampir pada bab 4, dapat diamati pengaruh parameter-parameter terhadap kualitas kompresi

Ambang batas (Threshold)

Dengan membandingkan kasus dengan ambang batas besar dan kecil, dapat diamati pengaruh berikut:

Threshold Kecil:

- Hanya blok dengan warna sangat seragam yang tidak dipecah (kualitas tinggi).
- Kualitas gambar lebih tinggi, tetapi ukuran file lebih besar (kurang kompresi).
- Contoh: Threshold = 5 → detail halus dipertahankan.

Threshold Besar:

- Blok dengan variasi warna lebih besar tetap tidak dipecah, (kualitas rendah)
- Ukuran file lebih kecil, tetapi kualitas menurun (kompresi tinggi).
- Contoh: Threshold = 50 → gambar lebih "blur".

Ukuran Blok Minimum

Dapat teramatinya bahwa nilai kecil meningkatkan detail tetapi menambah jumlah node (bandingkan ukuran minimum blok 4 dan 2). Ini cukup jelas karena ukuran minimum blok yang semakin kecil tentunya memperbolehkan pembagian daerah ke ukuran yang lebih kecil sehingga lebih banyak detail yang dipertahankan.

Metode Galat

Perhatikan bahwa metode/kalkulator galat MaxDiff dan Entropy memiliki performa lebih baik ketika ambang batas diperkecil, padahal metode lain seperti varians dan MAD dapat melakukan kompresi yang cukup baik dengan nilai ambang batas yang sama. Ini disebabkan karena sensitivitas pada perubahan lokal.

Tinjau metode MaxDiff, metode ini mengukur perbedaan maksimum antara nilai minimum dan maksimum di dalam suatu blok. Karena perhitungan hanya bergantung pada dua nilai ekstrem, satu perbedaan kecil yang signifikan (misalnya sebuah tepi atau noise) akan menghasilkan nilai error yang tinggi. Oleh karena itu, dengan menurunkan ambang batas, algoritma akan lebih selektif dalam menganggap suatu blok tidak homogen dan membagiannya lebih lanjut. Hasilnya, struktur blok yang dihasilkan akan lebih sesuai dengan detail lokal, yang bisa meningkatkan performa kompresi dalam mempertahankan struktur gambar.

Serupa halnya dengan Entropy, blok yang memiliki perbedaan intensitas yang kecil namun kompleks (misalnya, tekstur halus) dapat menghasilkan nilai entropi yang tinggi. Dengan ambang batas yang lebih rendah, perubahan kecil yang menunjukkan adanya ketidakhomogenan akan dideteksi dan blok akan dipecah untuk menangkap detail tersebut.

REFERENSI

- <https://medium.com/@tannerwyrk/quadtrees-for-image-processing-302536c95c00>
- <https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Stand ard-Deviation-from-Channels-Statistics/>
- [region-quadtree/ImageQuadTree.py at master · TannerYork/region-quadtree](https://region-quadtrees/region-quadtree/ImageQuadTree.py)
- <https://www.scitepress.org/papers/2013/42105/42105.pdf>
- [https://docs.oracle.com/javase/8/docs/api/java/awt/image/package-summary.h tml](https://docs.oracle.com/javase/8/docs/api/java/awt/image/package-summary.html)
- <https://www.color.org/chardata/rgb/bt601.xalter>
- <https://github.com/rtyley/animated-gif-lib-for-java>
- [Coding Challenge #98.2: Quadtree - Part 2](#)

LAMPIRAN

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 galat wajib	✓	
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan		✓
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran galat		✓
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8	Program dan laporan dibuat (kelompok) sendiri	✓	

pranala repository: https://github.com/xinuzo/Tucil2_10123083.git

