

**LAPORAN TUGAS KECIL 2**  
**MATA KULIAH IF2211 STRATEGI ALGORITMA**  
**KOMPRESI GAMBAR DENGAN METODE QUADTREE**



**Dosen Pengampu**  
Dr. Nur Ulfa Maulidevi, S.T, M.Sc.

**Disusun Oleh :**

Nadhif Radityo Nugroho 13523045

Adhimas Aryo Bimo 13523052

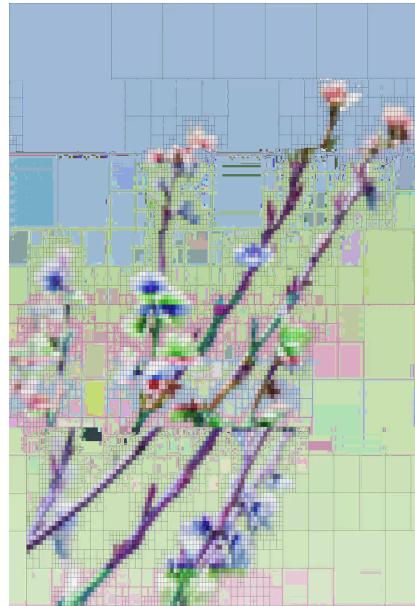
**SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA**  
**INSTITUT TEKNOLOGI BANDUNG**  
**APRIL 2025**

## DAFTAR ISI

<b>BAB I LATAR BELAKANG.....</b>	<b>3</b>
1.1. Metode Kompresi Quadtree.....	3
1.2. Perhitungan Pengukuran Metode Error.....	4
<b>BAB II DESKRIPSI DAN IMPLEMENTASI ALGORITMA.....</b>	<b>7</b>
2.1. Alur Berpikir.....	7
2.2. Cara Kerja Program.....	8
<b>BAB III SEKILAS KODE PROGRAM.....</b>	<b>13</b>
3.1. Struktur Program.....	13
3.2. Implementasi Program.....	14
3.3. Optimasi Program - Penyimpanan Efisien.....	25
3.4. Optimasi Program - Perhitungan Statistik Gambar.....	26
3.5. Optimasi Program - Threading.....	28
3.6. Bonus - Implementasi Structural Similarity Index (SSIM).....	29
3.7. Bonus - Output GIF.....	29
<b>BAB IV PERCOBAAN PROGRAM.....</b>	<b>30</b>
4.1. Eksperimen 1.....	30
4.2. Eksperimen 2.....	31
4.3. Eksperimen 3.....	32
4.4. Eksperimen 4.....	33
4.5. Eksperimen 5.....	35
4.6. Eksperimen 6.....	36
4.7. Eksperimen 7.....	37
<b>BAB V ANALISIS.....</b>	<b>39</b>
5.1. Analisis.....	39
5.2. Kompleksitas Algoritma.....	40
5.3. Kesimpulan.....	40
5.4. Saran.....	41
<b>BAB VI LAMPIRAN.....</b>	<b>42</b>
6.1. Tautan.....	42
6.2. Tabel Laporan.....	42

# BAB I LATAR BELAKANG

## 1.1. Metode Kompresi Quadtree



**Gambar 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.

## 1.2. Perhitungan Pengukuran Metode Error

Dalam menghitung seberapa besar perbedaan dalam satu blok gambar diperlukan metode pengukuran error untuk membatasi perbedaan tersebut. Pembatasan ini diperlukan untuk membatasi split tiap tree, jika error memenuhi batas nilai threshold tertentu, maka blok akan dibagi menjadi bagian yang lebih kecil. Dalam program ini terdapat 5 metode penghitungan galat pada blok, yakni berdasarkan variansi, *mean absolute deviation* (MAD), *max pixel difference* (MPD), Entropy, dan *structural similarity index measurement* (SSIM).

### a. Perhitungan Error Variansi

Untuk menghitung galat berdasarkan variansi, secara matematis dapat digunakan rumus sebagai berikut:

$$\begin{aligned}\sigma_c^2 &= \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2 \\ \sigma_{RGB}^2 &= \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}\end{aligned}$$

Dengan keterangan sebagai berikut:

- $\sigma_c^2$  = Variansi tiap kanal warna c (R, G, B) dalam satu blok  
 $P_{i,c}$  = Nilai piksel pada posisi i untuk kanal warna c  
 $\mu_c$  = Nilai rata-rata tiap piksel dalam satu blok  
N = Banyaknya piksel dalam satu blok

### b. Perhitungan Error Mean Absolute Deviation

Untuk menghitung galat berdasarkan *mean absolute deviation* (MAD), secara matematis dapat digunakan rumus sebagai berikut:

$$\begin{aligned}MAD_c &= \frac{1}{N} \sum_{i=1}^N |P_{i,c} - \mu_c| \\ MAD_{RGB} &= \frac{MAD_R + MAD_G + MAD_B}{3}\end{aligned}$$

Dengan keterangan sebagai berikut:

- $MAD_c$  = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok  
 $P_{i,c}$  = Nilai piksel pada posisi i untuk kanal warna c  
 $\mu_c$  = Nilai rata-rata tiap piksel dalam satu blok

N = Banyaknya piksel dalam satu blok

### c. Perhitungan Error Max Pixel Difference

Untuk menghitung galat berdasarkan *max pixel difference* (MPD), secara matematis dapat digunakan rumus sebagai berikut:

$$D_c = \max(P_{i,c}) - \min(P_{i,c})$$

$$D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

Dengan keterangan sebagai berikut:

$D_c$  = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok

$P_{i,c}$  = Nilai piksel pada posisi i untuk channel warna c

### d. Perhitungan Error Entropy

Untuk menghitung galat berdasarkan Entropy, secara matematis dapat digunakan rumus sebagai berikut:

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

Dengan keterangan sebagai berikut:

$H_c$  = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok

$P_c(i)$  = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna (R, G, B)

### e. Perhitungan Error Structural Similarity Index

Untuk menghitung galat berdasarkan *structural similarity index measurement* (SSIM), secara matematis dapat digunakan rumus sebagai berikut:

$$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$$

$$SSIM_{RGB} = w_R * SSIM_R + w_G * SSIM_G + w_B * SSIM_B$$

Dengan keterangan sebagai berikut:

$\mu_{x,c}$  = Rata-rata (mean) piksel pada blok pertama (misalnya: blok gambar asli) untuk kanal warna c

$\mu_{y,c}$  = Rata-rata (mean) piksel pada blok kedua (misalnya: blok hasil kompresi) untuk kanal warna c

$\sigma_{x,c}^2$  = Variansi dari blok x di kanal c

$\sigma_{y,c}^2$  = Variansi dari blok y di kanal c

$\sigma_{xy,c}$  = Kovarians antara blok x dan y di kanal c

$$C_1 = (K_1 * L)^2, \quad C_2 = (K_2 * L)^2$$

Dengan  $K_1 = 0.01$  dan  $K_2 = 0.03$ , dan L nilai maksimum pixel = 255

$$C_1 = (0.01 \cdot 255)^2 = 6.5025$$

$$C_2 = (0.03 \cdot 255)^2 = 58.5225$$

## BAB II DESKRIPSI DAN IMPLEMENTASI ALGORITMA

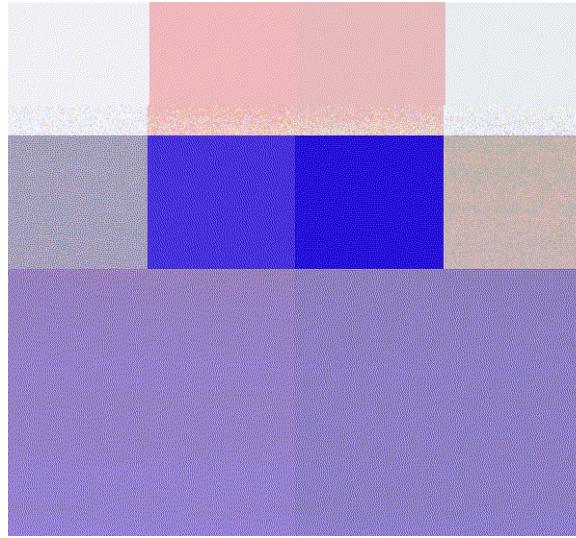
### 2.1. Alur Berpikir

Secara nalar, algoritma kompresi dalam laporan ini merupakan pendekatan kompresi gambar dengan menggunakan Quad Tree. Tiap-tiap wilayah yang terdapat dalam Quad Tree akan dibandingkan nilai errornya dengan referensi gambar asli. Apabila error tersebut masih dapat dianulir, maka tidak akan terjadi pembagian wilayah. Yang mana pendekatan konsep ini adalah inti utama kompresi, yaitu pengurangan detail fitur gambar yang dibawah toleransi. Sebaliknya, apabila error di atas ambang toleransi, maka akan dilakukan pembagian wilayah. Pada pendekatan algoritma Quad Tree pembagian wilayah akan dibagi menjadi 4 dengan masing-masing wilayah memiliki tinggi dan lebar yang sama (alignment akan diperlukan apabila kelipatan tinggi dan lebar dari gambar bukan kelipatan dari ukuran blok minimum).

Program yang diimplementasikan dalam laporan ini, mengikuti pendekatan konsep *divide and conquer* dalam penerapan Quad Tree. Secara singkat, program akan dimulai dengan penambahan wilayah root tree yaitu  $X=0$ ,  $Y=0$ ,  $W=\text{lebar gambar}$  dan,  $H=\text{tinggi gambar}$ . Kemudian pada langkah berikutnya, akan dihitung nilai error pada wilayah root tersebut. Apabila nilai error sudah di bawah ambang threshold, maka program akan berhenti. Sebaliknya, apabila nilai error masih di atas ambang threshold, maka akan dilakukan 4 pembagian wilayah ( $P$  adalah wilayah parent, asumsi bahwa dimensi gambar adalah pangkat dari 2):

1. Wilayah TL:  $X=P.X$ ,  $Y=P.Y$ ,  $W=P.W/2$ ,  $H=P.H/2$
2. Wilayah TR:  $X=P.X+P.W/2$ ,  $Y=P.Y$ ,  $W=P.W/2$ ,  $H=P.H/2$
3. Wilayah BL:  $X=P.X$ ,  $Y=P.Y+P.H/2$ ,  $W=P.W/2$ ,  $H=P.H/2$
4. Wilayah BR:  $X=P.X+P.W/2$ ,  $Y=P.Y+P.H/2$ ,  $W=P.W/2$ ,  $H=P.H/2$

Setelah pembagian wilayah-wilayah tersebut, langkah berikutnya adalah mengecek kembali dari tiap-tiap wilayah tersebut nilai errornya lalu dibandingkan dengan ambang threshold, dan apabila errornya masih di atas threshold maka akan dilakukan pembagian wilayah tersebut kembali.



**Gambar 2.** Proses Pembentukan Quadtree dalam Kompresi Gambar

(Sumber: [https://dahtah.github.io/imager/quadtrees\\_apple.gif](https://dahtah.github.io/imager/quadtrees_apple.gif))

## 2.2. Cara Kerja Program

Sebelum menjelaskan mengenai algoritma *divide and conquer*, perlu dikenalkan ADT Quad Tree yang dapat membagi wilayah secara bebas. Pada bagian implementasi lainnya, akan digunakan ADT ini untuk membantu proses pembagian wilayah sehingga *separation of concerns* tetap terjaga.

```
type QuadTreeNode :  
  <x : integer,  
   y : integer,  
   w : integer,  
   h : integer,  
   TL : integer,  
   TR : integer,  
   BL : integer,  
   BR : integer>  
  
{KAMUS}  
  tree : array of QuadTreeNode  
  alignX, alignY : integer  
  nodeCount : integer  
  
function createNode(INPUT x : integer, INPUT y : integer, INPUT w : integer,  
 INPUT h : integer) → integer  
  
{KAMUS}  
  idx : integer  
  
{ALGORITMA}  
  idx ← nodeCount  
  nodeCount ← nodeCount + 1
```

```

tree[idx].x ← x
tree[idx].y ← y
tree[idx].w ← w
tree[idx].h ← h
tree[idx].TL ← -1
tree[idx].TR ← -1
tree[idx].BL ← -1
tree[idx].BR ← -1
→ idx

procedure split(INPUT idx : integer)

{KAMUS}
    x, y, w, h : integer
    midX, midY : integer
    w1, w2, h1, h2 : integer

{ALGORITMA}
    x ← tree[idx].x
    y ← tree[idx].y
    w ← tree[idx].w
    h ← tree[idx].h
    midX ← ceil((x + w / 2) / alignX) * alignX
    midY ← ceil((y + h / 2) / alignY) * alignY

    if (midX ≤ x or midX ≥ x + w) then
        midX ← x + w div 2
    if (midY ≤ y or midY ≥ y + h) then
        midY ← y + h div 2

    w1 ← midX - x
    w2 ← x + w - midX
    h1 ← midY - y
    h2 ← y + h - midY

    if (tree[idx].TL = -1) then
        tree[idx].TL ← createNode(x, y, w1, h1)
    if (tree[idx].TR = -1) then
        tree[idx].TR ← createNode(midX, y, w2, h1)
    if (tree[idx].BL = -1) then
        tree[idx].BL ← createNode(x, midY, w1, h2)
    if (tree[idx].BR = -1) then
        tree[idx].BR ← createNode(midX, midY, w2, h2)

```

Selanjutnya akan diberikan *pseduocode* yang merupakan bagian penting dari program ini. Bagian ini merupakan bagian yang akan menentukan apakah suatu wilayah akan dibagi kembali dengan menerapkan perhitungan error.

```

type ImageQuadTreeCompressor :
    <image : ImageBuffer,
     imageStatistics : ImageStatistics,

```

```

quadTree : Boundary2DQuadTree,
quadTreeColors : array of integer,
quadTreeIndexQueue : array of integer>

procedure step() → boolean

{KAMUS}
node : integer
x, y, w, h : integer
shouldSplit : boolean
averageColor : integer
listCurrentNodes : array of integer { himpunan node yang sedang
diproses }
listNewNodes : array of integer { himpunan node hasil split }

{ALGORITMA}
listCurrentNodes ← quadTreeIndexQueue
listNewNodes ← []

for each node in listCurrentNodes do
    x ← quadTree.getBoundaryX(node)
    y ← quadTree.getBoundaryY(node)
    w ← quadTree.getBoundaryW(node)
    h ← quadTree.getBoundaryH(node)

    if (w ≤ 1 or h ≤ 1 or w ≤ minBlockSize or h ≤ minBlockSize) then
        shouldSplit ← false
    else
        shouldSplit ← controller.shouldSplit(image, imageStatistics, x,
y, w, h)
        averageColor ← imageStatistics.getAverageColor(x, y, w, h)

        if (shouldSplit) then
            quadTree.split(node)
            listNewNodes ← listNewNodes ∪ {
                quadTree.getIndexTL(node),
                quadTree.getIndexTR(node),
                quadTree.getIndexBL(node),
                quadTree.getIndexBR(node)
            }
            quadTreeColors[node] = averageColor { simpan hasil rata-rata untuk
kompresi }

        if (listNewNodes is not empty) then
            quadTreeIndexQueue ← listNewNodes { update node yang akan diproses
pada iterasi berikutnya }
            → true
        else
            → false

```

Sebelum prosedur *step()* dimulai perlu ada inisialisasi wilayah root pada Quad Tree. Buat satu node akar QuadTree yang mencakup seluruh dimensi gambar (width × height dengan x=0,

$y=0$ ). Tambahkan indeks node ke dalam *quadTreeIndexQueue*. Secara singkat, proses *divide and conquer* dapat dijabarkan sebagai berikut:

1. Ambil semua node yang akan diproses dengan menyalin seluruh node dari *quadTreeIndexQueue* ke *listCurrentNodes*. Lalu, Kosongkan *quadTreeIndexQueue* untuk digunakan pada iterasi berikutnya.
2. Dilakukan dalam beberapa iterasi (step-by-step). Di setiap langkah:
  - a. Untuk setiap node, ambil lokasi dan ukuran blok ( $x$ ,  $y$ ,  $w$ ,  $h$ ). Lalu, evaluasi homogenitas blok dengan metode *shouldSplit()*: Jika blok terlalu kecil ( $\leq minBlockSize$ ) atau tidak layak dibagi, maka dianggap leaf (daun). Jika blok masih heterogen (warna bervariasi signifikan atau error yang di atas ambang *threshold*), maka harus dipecah lebih lanjut.
  - b. Hitung rata-rata warna. Jika blok dianggap cukup homogen, maka warna rata-rata blok dihitung untuk mewakili warna keseluruhan area.
  - c. Lakukan pembagian wilayah jika perlu dengan membagi node menjadi empat kuadran: *TL (Top-Left)*, *TR (Top-Right)*, *BL (Bottom-Left)*, *BR (Bottom-Right)*. Tambahkan keempat node hasil split ke dalam antrian proses (*quadTreeIndexQueue*) untuk iterasi selanjutnya. Pembagian 4 wilayah digunakan rumus sebagai berikut ( $P$  adalah wilayah parent, asumsi bahwa dimensi gambar adalah pangkat dari 2):
    - i. Wilayah TL:  $X=P.X$ ,  $Y=P.Y$ ,  $W=P.W/2$ ,  $H=P.H/2$
    - ii. Wilayah TR:  $X=P.X+P.W/2$ ,  $Y=P.Y$ ,  $W=P.W/2$ ,  $H=P.H/2$
    - iii. Wilayah BL:  $X=P.X$ ,  $Y=P.Y+P.H/2$ ,  $W=P.W/2$ ,  $H=P.H/2$
    - iv. Wilayah BR:  $X=P.X+P.W/2$ ,  $Y=P.Y+P.H/2$ ,  $W=P.W/2$ ,  $H=P.H/2$
3. Proses akan terus diulang (dipanggil lagi dengan *step()*) selama masih ada node yang perlu diproses di antrian (*quadTreeIndexQueue*). Jika sudah tidak ada node untuk diproses (antrian kosong), maka proses pembentukan pohon QuadTree selesai.

Meskipun implementasi menggunakan struktur rekursif implisit, konsep *divide and conquer* tetap berlaku. Pada tiap-tiap queue terdapat analogi perbandingan yang sama, yaitu terdapat: *Divide* yaitu memecah blok menjadi empat bagian, lalu *conquer* yaitu mengevaluasi dan memproses masing-masing bagian secara rekursif (melalui iterasi), dan terakhir *combine* yaitu menyimpan hasil representasi warna (rata-rata) dan struktur pohon di *quadTree* untuk digunakan dalam visualisasi atau kompresi akhir.

Bagian terakhir adalah rekonstruksi ulang struktur Quad Tree yang telah dikompres menjadi sebuah gambar yang dapat dibuka secara langsung. Secara singkat, prosesnya adalah sebagai berikut:

```

function getCompressedImage() → image

{KAMUS}
    compressedImage : BufferedImage
    graphics : Graphics2D
    queue : array of integer
    quadTreeIndex : integer
    x, y, w, h : integer
    color : integer
    indexTL, indexTR, indexBL, indexBR : integer

{ALGORITMA}
    compressedImage ← BufferedImage(image.width, image.height)
    graphics ← compressedImage.graphics

    queue[0] ← 0
    while (length(queue) > 0) do
        quadTreeIndex ← queue.takeFirst()

        x ← quadTree.getBoundaryX(quadTreeIndex)
        y ← quadTree.getBoundaryY(quadTreeIndex)
        w ← quadTree.getBoundaryW(quadTreeIndex)
        h ← quadTree.getBoundaryH(quadTreeIndex)
        color ← quadTreeColors[quadTreeIndex]

        graphics.setColor(color)
        graphics.fillRect(x, y, w, h)

        indexTL ← quadTree.getIndexTL(quadTreeIndex)
        indexTR ← quadTree.getIndexTR(quadTreeIndex)
        indexBL ← quadTree.getIndexBL(quadTreeIndex)
        indexBR ← quadTree.getIndexBR(quadTreeIndex)

        if (indexTL != -1) then
            add(queue, indexTL)
        if (indexTR != -1) then
            add(queue, indexTR)
        if (indexBL != -1) then
            add(queue, indexBL)
        if (indexBR != -1) then
            add(queue, indexBR)

    → compressedImage

```

## BAB III SEKILAS KODE PROGRAM

### 3.1. Struktur Program

Berikut merupakan struktur file dari program:

```
Tucil2_13523045_13523052/
├── README.md                                # Deskripsi dan petunjuk penggunaan
program
├── build/                                    # Hasil kompilasi Gradle
│   ├── classes/                             # File .class hasil kompilasi
│   └── libs/                                # File .jar hasil build
│       └── Tucil2_13523045_13523052.jar
├── build.gradle.kts                         # Konfigurasi build Gradle (Kotlin
DSL)
├── docs/                                     # Dokumentasi tugas
│   ├── Tucil2-Stima-2025.pdf
│   └── Tucil2_13523045_13523052.pdf
├── gradle/                                   # Konfigurasi internal Gradle
│   ├── libs.versions.toml
│   └── wrapper/
│       ├── gradle-wrapper.jar
│       └── gradle-wrapper.properties
├── gradle.properties                         # Properti global Gradle
├── gradlew                                    # Script build Gradle (Unix)
├── gradlew.bat                               # Script build Gradle (Windows)
├── settings.gradle.kts                      # Setting proyek Gradle
└── src/
    ├── main/
    │   └── java/
    │       └── tucil2_13523045_13523052/
    │           ├── Boundary2DQuadTree.java
    │           ├── GifSequenceWriter.java
    │           ├── ImageBuffer.java
    │           ├── ImageQuadTreeCompressor.java
    │           ├── ImageStatistics.java
    │           ├── Main.java
    │           └── Utils.java
    └── test/
        ├── java/
        │   └── tucil2_13523045_13523052/
        │       └── test/
        └── resources/
└── test/
    ├── eksperimen1/
    ├── eksperimen2/
    └── eksperimen3/
```

```

    └── eksperimen4/
    └── eksperimen5/
    └── eksperimen6/
    └── eksperimen7/
    └── .gitignore           # File untuk mengecualikan file saat
    commit
    └── .gitattributes        # Konfigurasi git tambahan
    └── .vscode/

```

Berikut merupakan penjelasan file-file utama:

File	Penggunaan
Boundary2DQuadTree.java	Merupakan <i>class Abstract Data Type</i> (ADT) Quad Tree yang dapat membagi suatu wilayah sembarang.
GifSequenceWriter.java	Merupakan <i>class</i> utilitas untuk membuat file GIF. CC By-SA 3.0, Elliot Kroo @ 2009-04-25. <a href="#">Sumber</a> .
ImageBuffer.java	Merupakan <i>class</i> utilitas untuk menyimpan dan membaca piksel dari suatu gambar. Modifikasi tidak diperbolehkan dalam <i>class</i> ini, hanya baca.
ImageQuadTreeCompressor.java	Merupakan <i>class</i> yang memiliki logika utama dalam perhitungan error, queueing Quad Tree, dan pembuatan gambar yang telah dikompres.
ImageStatistics.java	Merupakan <i>class</i> utilitas yang dapat memberikan statistik seperti <i>average</i> , <i>variance</i> , dan lainnya. Memiliki prosedur pre komputasi di dalamnya untuk mengefisienkan waktu.
Main.java	Logika utama untuk <i>Command Line Interface</i> (CLI).
Utils.java	<i>Class</i> utilitas yang memiliki beberapa method

untuk digunakan di berbagai tempat.

### 3.2. Implementasi Program

```
Boundary2DQuadTree.java dde0b77c78362ef014330c7caa6dae22f27d1a6d

package tucil2_13523045_13523052;

import java.util.ArrayList;
import java.util.List;

import org.eclipse.collections.api.factory.primitive.IntLists;
import org.eclipse.collections.api.list.primitive.MutableIntList;
import org.eclipse.collections.impl.list.mutable.primitive.IntArrayList;

public class Boundary2DQuadTree {
    protected final int alignX;
    protected final int alignY;
    protected final MutableIntList indices;
    protected final MutableIntList boundaries;
    protected final MutableIntList skipped;

    public Boundary2DQuadTree(int x, int y, int width, int height, int
alignX, int alignY) {
        if(x % alignX != 0 || y % alignY != 0)
            throw new Error("Origin is not aligned");
        this.alignX = alignX;
        this.alignY = alignY;
        this.indices = IntLists.mutable.empty();
        this.boundaries = IntLists.mutable.empty();
        this.skipped = IntLists.mutable.empty();
        newTree(x, y, width, height);
    }

    public int getIndexTL(int index) {
        return indices.get(index * 4 + 0);
    }
    public int getIndexTR(int index) {
        return indices.get(index * 4 + 1);
    }
    public int getIndexBL(int index) {
        return indices.get(index * 4 + 2);
    }
    public int getIndexBR(int index) {
        return indices.get(index * 4 + 3);
    }
    protected void setIndexTL(int index, int value) {
        indices.set(index * 4 + 0, value);
    }
    protected void setIndexTR(int index, int value) {
        indices.set(index * 4 + 1, value);
    }
}
```

```

    }
    protected void setIndexBL(int index, int value) {
        indices.set(index * 4 + 2, value);
    }
    protected void setIndexBR(int index, int value) {
        indices.set(index * 4 + 3, value);
    }
    public int getBoundaryX(int index) {
        return boundaries.get(index * 4 + 0);
    }
    public int getBoundaryY(int index) {
        return boundaries.get(index * 4 + 1);
    }
    public int getBoundaryW(int index) {
        return boundaries.get(index * 4 + 2);
    }
    public int getBoundaryH(int index) {
        return boundaries.get(index * 4 + 3);
    }
    protected void setBoundaryX(int index, int value) {
        boundaries.set(index * 4 + 0, value);
    }
    protected void setBoundaryY(int index, int value) {
        boundaries.set(index * 4 + 1, value);
    }
    protected void setBoundaryW(int index, int value) {
        boundaries.set(index * 4 + 2, value);
    }
    protected void setBoundaryH(int index, int value) {
        boundaries.set(index * 4 + 3, value);
    }

    public int getNodeCount() {
        return indices.size() / 4 - skipped.size();
    }
    public int getMaxDepth() {
        return getMaxDepthRecursive(0);
    }
    private int getMaxDepthRecursive(int index) {
        int indexTL = getIndexTL(index);
        int indexTR = getIndexTR(index);
        int indexBL = getIndexBL(index);
        int indexBR = getIndexBR(index);
        int maxChildDepth = Math.max(
            Math.max(indexTL != -1 ? getMaxDepthRecursive(indexTL) :
0, indexTR != -1 ? getMaxDepthRecursive(indexTR) : 0),
            Math.max(indexBL != -1 ? getMaxDepthRecursive(indexBL) :
0, indexBR != -1 ? getMaxDepthRecursive(indexBR) : 0)
        );
        return maxChildDepth + 1;
    }
    public int getTotalLeafArea() {
        int total = 0;

```

```

var queue = IntLists.mutable.empty();
queue.add(0);
while(queue.size() > 0) {
    int index = queue.removeAtIndex(0);
    int indexTL = getIndexTL(index);
    int indexTR = getIndexTR(index);
    int indexBL = getIndexBL(index);
    int indexBR = getIndexBR(index);
    if(indexTL == -1 && indexTR == -1 && indexBL == -1 &&
indexBR == -1) {
        int w = getBoundaryW(index);
        int h = getBoundaryH(index);
        total += w * h;
    } else {
        if(indexTL != -1) queue.add(indexTL);
        if(indexTR != -1) queue.add(indexTR);
        if(indexBL != -1) queue.add(indexBL);
        if(indexBR != -1) queue.add(indexBR);
    }
}
return total;
}

protected int newTree(int x, int y, int w, int h) {
    int index = indices.size() / 4;
    if(skipped.size() > 0) {
        index = skipped.getLast();
        skipped.removeAtIndex(skipped.size() - 1);
    } else {
        Utils.growMutableIntList(indices, indices.size() + 4);
        Utils.growMutableIntList(boundaries, boundaries.size() +
4);
    }
    setIndexTL(index, -1);
    setIndexTR(index, -1);
    setIndexBL(index, -1);
    setIndexBR(index, -1);
    setBoundaryX(index, x);
    setBoundaryY(index, y);
    setBoundaryW(index, w);
    setBoundaryH(index, h);
    return index;
}
protected void deleteTree(int index) {
    skipped.add(index);
    setIndexTL(index, -1);
    setIndexTR(index, -1);
    setIndexBL(index, -1);
    setIndexBR(index, -1);
    setBoundaryX(index, -1);
    setBoundaryY(index, -1);
    setBoundaryW(index, -1);
    setBoundaryH(index, -1);
}

```

```

    }
    public void split(int index) {
        int x = getBoundaryX(index);
        int y = getBoundaryY(index);
        int w = getBoundaryW(index);
        int h = getBoundaryH(index);
        int midX = ((x + w / 2 + alignX - 1) / alignX) * alignX;
        int midY = ((y + h / 2 + alignY - 1) / alignY) * alignY;
        if(midX <= x || midX >= x + w) midX = x + w / 2;
        if(midY <= y || midY >= y + h) midY = y + h / 2;
        int w1 = midX - x;
        int w2 = x + w - midX;
        int h1 = midY - y;
        int h2 = y + h - midY;
        if(getIndexTL(index) == -1)
            setIndexTL(index, newTree(x, y, w1, h1));
        if(getIndexTR(index) == -1)
            setIndexTR(index, newTree(midX, y, w2, h1));
        if(getIndexBL(index) == -1)
            setIndexBL(index, newTree(x, midY, w1, h2));
        ) if(getIndexBR(index) == -1) setIndexBR(index, newTree(midX, midY, w2,
(          qqT)                                         exT
                                         C

```

## ImageQuadTreeCompressor.java dde0b77c78362ef014330c7caa6dae22f27d1a6d

```
package tucil2_13523045_13523052;

import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.concurrent.CompletableFuture;

import org.eclipse.collections.api.factory.primitive.IntLists;
import org.eclipse.collections.api.list.primitive.MutableIntList;

public class ImageQuadTreeCompressor {
    public static abstract class Controller {
        public final double threshold;
        public Controller(double threshold) {
            this.threshold = threshold;
        }
        public abstract boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h);
    }

    public static class Variance extends Controller {
        public Variance(double threshold) {
            super(threshold);
        }
        @Override
        public boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h) {
            int pixelCount = w * h;
            double meanR = imageStatistics.getAverageR(x, y, w,
h);
            double meanG = imageStatistics.getAverageG(x, y, w,
h);
            double meanB = imageStatistics.getAverageB(x, y, w,
h);

            // Calculate variance
            double varianceR = 0;
            double varianceG = 0;
            double varianceB = 0;
            for(int j = y; j < y + h; j++) {
                int et=g ; < + ; ++) {
                    varianceR += Math.pow(image.getPixel(j, g) - meanR, 2);
                    varianceG += Math.pow(image.getPixel(j, g) - meanG, 2);
                    varianceB += Math.pow(image.getPixel(j, g) - meanB, 2);
                }
            }
            return varianceR > threshold || varianceG > threshold || varianceB > threshold;
        }
    }
}
```

```

        double varRGB = (varianceR + varianceG + varianceB)
/ 3.0;

        // Check if variance is above threshold
        return varRGB > threshold;
    }
}

public static class MeanAbsoluteDeviation extends Controller {
    public MeanAbsoluteDeviation(double threshold) {
        super(threshold);
    }
    @Override
    public boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h) {
        int pixelCount = w * h;
        double meanR = imageStatistics.getAverageR(x, y, w,
h);
        double meanG = imageStatistics.getAverageG(x, y, w,
h);
        double meanB = imageStatistics.getAverageB(x, y, w,
h);

        // Calculate Mean Absolute Deviation
        double madR = 0;
        double madG = 0;
        double madB = 0;
        for(int j = y; j < y + h; j++) {
            for(int i = x; i < x + w; i++) {
                madR += Math.abs(image.getRed(i, j) -
meanR);
                madG += Math.abs(image.getGreen(i, j))
                madB += Math.abs(image.getBlue(i, j) -
meanB);
            }
        }
        madR /= pixelCount;
        madG /= pixelCount;
        madB /= pixelCount;
        double madRGB = (madR + madG + madB) / 3.0;

        // Check if MAD is above threshold
        return madRGB > threshold;
    }
}

public static class MaxPixelDifference extends Controller {
    public MaxPixelDifference(double threshold) {
        super(threshold);
    }
    @Override
    public boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h) {
        int maxR = 0;

```

```

        int maxG = 0;
        int maxB = 0;
        int minR = 255;
        int minG = 255;
        int minB = 255;

        // Calculate max and min values
        for(int j = y; j < y + h; j++) {
            for(int i = x; i < x + w; i++) {
                int r = image.getRed(i, j);
                int g = image.getGreen(i, j);
                int b = image.getBlue(i, j);
                maxR = Math.max(maxR, r);
                maxG = Math.max(maxG, g);
                maxB = Math.max(maxB, b);
                minR = Math.min(minR, r);
                minG = Math.min(minG, g);
                minB = Math.min(minB, b);
            }
        }
        double diffR = maxR - minR;
        double diffG = maxG - minG;
        double diffB = maxB - minB;
        double diffRGB = (diffR + diffG + diffB) / 3.0;

        // Calculate the maximum pixel difference
        return diffRGB > threshold;
    }
}

public static class Entropy extends Controller {
    public Entropy(double threshold) {
        super(threshold);
    }
    @Override
    public boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h) {
        int pixelCount = w * h;
        int[] histR = new int[256];
        int[] histG = new int[256];
        int[] histB = new int[256];

        // Compute histogram
        for(int j = y; j < y + h; j++) {
            for(int i = x; i < x + w; i++) {
                int r = image.getRed(i, j);
                int g = image.getGreen(i, j);
                int b = image.getBlue(i, j);
                histR[r]++;
                histG[g]++;
                histB[b]++;
            }
        }
    }
}

```

```

        // Compute Entropy per channel
        double entropyR = 0;
        double entropyG = 0;
        double entropyB = 0;
        for(int i = 0; i < 256; i++) {
            if(histR[i] > 0) {
                double p = (double) histR[i] /
pixelCount;
                entropyR -= p * Math.log(p) /
Math.log(2);
            }
            if(histG[i] > 0) {
                double p = (double) histG[i] /
pixelCount;
                entropyG -= p * Math.log(p) /
Math.log(2);
            }
            if(histB[i] > 0) {
                double p = (double) histB[i] /
pixelCount;
                entropyB -= p * Math.log(p) /
Math.log(2);
            }
        }
        double entropy = (entropyR + entropyG + entropyB) /
3.0;
        return entropy > threshold;
    }
}

public static class StructuralSimilarityIndex extends Controller
{
    private static final double C1 = 6.5025;
    private static final double C2 = 58.5225;

    public StructuralSimilarityIndex(double threshold) {
        super(threshold);
    }

    private double computeSSIM(double muX, double sigmaX2,
double muY, double sigmaXY) {
        double numerator = (2 * muX * muY + C1) * (2 *
sigmaXY + C2);
        double denominator = (muX * muX + muY * muY + C1) *
(sigmaX2 + sigmaXY + C2);
        return denominator == 0 ? 1 : numerator /
denominator;
    }
    @Override
    public boolean shouldSplit(ImageBuffer image,
ImageStatistics imageStatistics, int x, int y, int w, int h){
        int pixelCount = w * h;
        double meanR = imageStatistics.getAverageR(x, y, w,

```

```

    h);
        double meanG = imageStatistics.getAverageG(x, y, w,
    h);
        double meanB = imageStatistics.getAverageB(x, y, w,
    h);

        // Calculate variance and covariance
        double varR = 0;
        double varG = 0;
        double varB = 0;
        for(int j = y; j < y + h; j++) {
            for(int i = x; i < x + w; i++) {
                varR += Math.pow(image.getRed(i, j) -
meanR, 2);
                varG += Math.pow(image.getGreen(i, j) -
meanG, 2);
                varB += Math.pow(image.getBlue(i, j) -
meanB, 2);
            }
        }
        varR /= pixelCount;
        varG /= pixelCount;
        varB /= pixelCount;

        // Karena membandingkan satu blok gambar dengan
        rata-ratanya sendiri maka, kovarians = 0
        double ssimR = computeSSIM(meanR, varR, meanR, 0);
        double ssimG = computeSSIM(meanG, varG, meanG, 0);
        double ssimB = computeSSIM(meanB, varB, meanB, 0);
        double ssim = (ssimR + ssimG + ssimB) / 3.0;

        // Check if SSIM too different from its average
        return ssim < threshold;
    }
}

protected final ImageBuffer image;
protected final ImageStatistics imageStatistics;
protected final Controller controller;
protected final int minBlockSize;
protected final Boundary2DQuadTree quadTree;
protected final MutableList quadTreeColors;
protected final MutableList quadTreeIndexQueue;

protected final BufferedImage compressionImage;
protected final Graphics2D compressionGraphics;
protected final MutableList drawCompressionQueue;

public ImageQuadTreeCompressor(ImageBuffer image, Controller
controller, int minBlockSize) {
    this.image = image;
    this.imageStatistics = new ImageStatistics(image, minBlockSize,

```

```

minBlockSize);
        this.controller = controller;
        this.minBlockSize = minBlockSize;
        this.quadTree = new Boundary2DQuadTree(0, 0, image.getWidth(),
image.getHeight(), minBlockSize, minBlockSize);
        this.quadTreeColors = IntLists.mutable.empty();
        this.quadTreeIndexQueue = IntLists.mutable.empty();
        this.quadTreeIndexQueue.add(0);

        this.compressionImage = new BufferedImage(image.getWidth(),
image.getHeight(), BufferedImage.TYPE_INT_ARGB);
        this.compressionGraphics =
this.compressionImage.createGraphics();
        this.drawCompressionQueue = IntLists.mutable.empty();
    }

    public boolean step() {
        MutableIntList queueCopy = quadTreeIndexQueue.toList();
        quadTreeIndexQueue.clear();
        boolean[] shouldSplitResults = new boolean[queueCopy.size()];
        int[] averageColorResults = new int[queueCopy.size()];
        int futureChunk = 64;
        int futureGroup = (queueCopy.size() + futureChunk - 1) /
futureChunk;
        CompletableFuture<?>[] futures = new
CompletableFuture[futureGroup];
        for(int i = 0; i < futureGroup; i++) {
            int startQueue = i * futureChunk;
            int endQueue = Math.min((i + 1) * futureChunk,
queueCopy.size());
            futures[i] = CompletableFuture.runAsync(() -> {
                for(int j = startQueue; j < endQueue; j++) {
                    int quadTreeIndex = queueCopy.get(j);
                    int x =
quadTree.getBoundaryX(quadTreeIndex);
                    int y =
quadTree.getBoundaryY(quadTreeIndex);
                    int w =
quadTree.getBoundaryW(quadTreeIndex);
                    int h =
quadTree.getBoundaryH(quadTreeIndex);
                    shouldSplitResults[j] = w <= 1 || h <= 1 ||
w <= minBlockSize || h <= minBlockSize ?
false : controller.shouldSplit(image,
imageStatistics, x, y, w, h);
                    averageColorResults[j] =
imageStatistics.getAverageColor(x, y, w, h);
                }
            });
        }
        for(int i = 0; i < futureGroup; i++) {
            int startQueue = i * futureChunk;
            int endQueue = Math.min((i + 1) * futureChunk,

```

```

queueCopy.size());
        futures[i].join();
        for(int j = startQueue; j < endQueue; j++) {
            int quadTreeIndex = queueCopy.get(j);
            boolean shouldSplit = shouldSplitResults[j];
            int averageColor = averageColorResults[j];
            if(shouldSplit) {
                quadTree.split(quadTreeIndex);

quadTreeIndexQueue.add(quadTree.getIndexTL(quadTreeIndex));
quadTreeIndexQueue.add(quadTree.getIndexTR(quadTreeIndex));
quadTreeIndexQueue.add(quadTree.getIndexBL(quadTreeIndex));
quadTreeIndexQueue.add(quadTree.getIndexBR(quadTreeIndex));
}
            Utils.growMutableIntList(quadTreeColors,
quadTreeIndex + 1);
            quadTreeColors.set(quadTreeIndex, averageColor);
drawCompressionQueue.add(quadTreeIndex);
        }
    }
    return quadTreeIndexQueue.size() > 0;
}
public BufferedImage getCompressedImage() {
    drawCompressionQueue.reverseThis();
    while(drawCompressionQueue.size() > 0) {
        int quadTreeIndex = drawCompressionQueue.getLast();

drawCompressionQueue.removeAtIndex(drawCompressionQueue.size() - 1);
        int x = quadTree.getBoundaryX(quadTreeIndex);
        int y = quadTree.getBoundaryY(quadTreeIndex);
        int w = quadTree.getBoundaryW(quadTreeIndex);
        int h = quadTree.getBoundaryH(quadTreeIndex);
        int color = quadTreeColors.get(quadTreeIndex);
        compressionGraphics.setColor(new Color(color, true));
        compressionGraphics.fillRect(x, y, w, h);
    }
    return compressionImage;
}

public int getNodeCount() {
    return quadTree.getNodeCount();
}
public int getTreeDepth() {
    return quadTree.getMaxDepth();
}
public int getTotalLeafArea() {
    return quadTree.getTotalLeafArea();
}
}

```

### 3.3. Optimasi Program - Penyimpanan Efisien

Strategi representasi Quad Tree dinilai sangat penting dalam pengimplementasian program ini. Pemilihan struktur yang salah dapat membuat program berjalan lama dan terlihat tidak responsif. Implementasi naif yang menggunakan *recursive references* dapat membuat performansi turun karena alamatnya yang berjauhan sehingga lokalitas tidak dapat dijamin.

Dalam implementasi program, digunakan suatu array satu dimensi yang dapat mendeskripsikan Quad Tree. Suatu *node* Quad Tree dapat dinyatakan dengan identifier uniknya, dalam hal ini akan digunakan *incrementing integer*. Lalu, properti seperti indeks *child nodes*, *boundaries*, dan warna block, dapat disimpan dalam suatu array independen dengan identifier node.

Child Nodes Array, Size: NodeCount * 4			
Index TL	Index TR	Index BL	Index BR
Index * 4 + 0	Index * 4 + 1	Index * 4 + 2	Index * 4 + 3
Catatan: Tiap-tiap indeks child akan mengacu ke indeks node Quad Tree lainnya.			

Node Boundaries Array, Size: NodeCount * 4			
X	Y	W	H
Index * 4 + 0	Index * 4 + 1	Index * 4 + 2	Index * 4 + 3

Color: Size: NodeCount			
R	G	B	A
(get(index) >> 16) & 0xFF	(get(index) >> 8) & 0xFF	get(index) & 0xFF	(get(index) >> 24) & 0xFF

### 3.4. Optimasi Program - Perhitungan Statistik Gambar

Dikarenakan terdapat perhitungan rerata yang sering terhadap suatu blok dalam gambar, maka akan lebih baik menggunakan strategi *integral image*. Strategi *integral image* adalah suatu teknik yang dapat menghitung jumlah suatu wilayah 2D yang nantinya dapat dibagi dengan jumlah piksel dalam wilayah sehingga akan mendapatkan rerata dalam wilayah tersebut.

Penggunaan strategi ini mengorbankan efisiensi ruang untuk memperoleh efisiensi waktu. Strategi ini memerlukan suatu array yang isinya telah dihitung sebelumnya, sehingga pada saat

ingin mencari jumlah suatu wilayah dapat dijalankan dengan kompleksitas  $O(1)$ . Dalam implementasi program, terdapat beberapa strategi penyimpanan *integral image* ini. Hal ini dibuat agar menyeimbangkan presisi dan ruang memori untuk gambar yang lebih besar.

Optimasi juga dilakukan apabila minimum block size besar. Artinya, setiap wilayah Quad Tree akan selalu *aligned* dengan minimum block, sehingga hanya perlu menyimpan semua integral untuk koordinat yang *aligned* dengan minimum block.

ImageStatistics.java dde0b77c78362ef014330c7caa6dae22f27d1a6d L202-233, L251-L268

```
public IntIntegral(ImageBuffer image, int band, int blockX, int blockY) {
    this.width = image.getWidth();
    this.height = image.getHeight();
    this.blockX = blockX;
    this.blockY = blockY;
    this.gridWidth = (width + blockX - 1) / blockX;
    this.gridHeight = (height + blockY - 1) / blockY;
    this.band = band;
    this.values = new int[gridHeight * gridWidth];
    for(int gy = 0; gy < gridHeight; gy++) {
        for(int gx = 0; gx < gridWidth; gx++) {
            int xStart = gx * blockX;
            int yStart = gy * blockY;
            int xEnd = Math.min(xStart + blockX, width);
            int yEnd = Math.min(yStart + blockY, height);
            int blockValue = 0;
            for(int y = yStart; y < yEnd; y++) {
                for(int x = xStart; x < xEnd; x++) {
                    blockValue += band == 0 ? image.getRed(x, y)
                                           : band == 1 ? image.getGreen(x, y) :
                                           band == 2 ? image.getBlue(x, y) :
                                           band == 3 ? image.getAlpha(x, y) :
                                           0;
                }
            }
            int above = gy > 0 ? values[(gy - 1) * gridWidth + gx] :
            0;
            int left = gx > 0 ? values[gy * gridWidth + gx - 1] : 0;
            int diag = gy > 0 && gx > 0 ? values[(gy - 1) * gridWidth +
            gx - 1] : 0;
            values[gy * gridWidth + gx] = blockValue + above + left -
            diag;
        }
    }
}
```

```

@Override
public long getIntegral(int x, int y, int w, int h) {
    if(x < 0 || x + w > width || y < 0 || y + h > height || w <= 0 || h <= 0)
        throw new IndexOutOfBoundsException();
    if(x + w == width) w = gridWidth * blockX - x;
    if(y + h == height) h = gridHeight * blockY - y;
    if(x % blockX != 0 || y % blockY != 0 || (x + w) % blockX != 0 || (y + h) % blockY != 0)
        throw new IllegalArgumentException("Coordinates must align with stride boundaries");
    x /= blockX;
    y /= blockY;
    w /= blockX;
    h /= blockY;
    int A = values[(y + h - 1) * gridWidth + x + w - 1];
    int B = x > 0 ? values[(y + h - 1) * gridWidth + x - 1] : 0;
    int C = y > 0 ? values[(y - 1) * gridWidth + x + w - 1] : 0;
    int D = x > 0 && y > 0 ? values[(y - 1) * gridWidth + x - 1] : 0;
    return (long) A - B - C + D;
}

```

### 3.5. Optimasi Program - Threading

Pemanfaatan *multiprocessing* dalam program ini terdapat pada bagian penyimpanan file dan perhitungan pre komputasi pada *class ImageStatistics*. Sebenarnya, pemanfaatan *divide and conquer* disini dapat dimanfaatkan dengan *threading* juga. Dengan lokalitas data tanpa pengaksesan bagian wilayah data lain, penggunaan *threading* sangat cocok karena tidak akan memerlukan sinkronisasi. Meskipun pada awal implementasi program memiliki rencana ini, pada saat profiling performance dibuktikan bahwa *bottleneck* sebenarnya terdapat di bagian encode gambar. Lalu, ditunjukkan kembali bahwa strategi optimasi penyimpanan efisien merupakan hal paling berpengaruh pada performa langkah *divide and conquer*.

Main.java dde0b77c78362ef014330c7caa6dae22f27d1a6d L264-L280

```

saveFutures.add(CompletableFuture.supplyAsync(() -> {
    try {
        long saveStart = System.nanoTime();
        ImageIO.write(pair.getOne(), format, finalImageFile);
        long saveDuration = System.nanoTime() - saveStart;
        saveTotalDuration.addAndGet(saveDuration);
    } catch(IOException e) {
        throw new RuntimeException(e);
    } finally {
        try {
            bufferImageQueue.put(pair);
        } catch(InterruptedException e) {

```

```

        throw new RuntimeException(e);
    }
}
return null;
}, Utils.executorService));

```

ImageStatistics.java dde0b77c78362ef014330c7caa6dae22f27d1a6d L14-L26

```

public ImageStatistics(ImageBuffer image, int integralBlockX, int
integralBlockY) {
    this.image = image;
    this.integralBlockX = integralBlockX;
    this.integralBlockY = integralBlockY;
    CompletableFuture<Integral> futureIntegralR =
CompletableFuture.supplyAsync(() -> constructIntegral(image, 0,
integralBlockX, integralBlockY));
    CompletableFuture<Integral> futureIntegralG =
CompletableFuture.supplyAsync(() -> constructIntegral(image, 1,
integralBlockX, integralBlockY));
    CompletableFuture<Integral> futureIntegralB =
CompletableFuture.supplyAsync(() -> constructIntegral(image, 2,
integralBlockX, integralBlockY));
    CompletableFuture<Integral> futureIntegralA =
CompletableFuture.supplyAsync(() -> constructIntegral(image, 3,
integralBlockX, integralBlockY));
    this.integralR = futureIntegralR.join();
    this.integralG = futureIntegralG.join();
    this.integralB = futureIntegralB.join();
    this.integralA = futureIntegralA.join();
}

```

### 3.6. Bonus - Implementasi Structural Similarity Index (SSIM)

Merupakan perhitungan metrik similaritas berdasarkan warna blok yang flat dengan detil struktur gambar original. Dengan menggunakan flag `'-m SSIM` atau `"--method SSIM` maka program secara otomatis menggunakan SSIM sebagai konsiderasi pembagian wilayah.

```

java.exe -jar Tucil2_13523045_13523052.jar -i .jpg1.jpg -m SSIM
-t 200 -b 80 -o out

```

### 3.7. Bonus - Output GIF

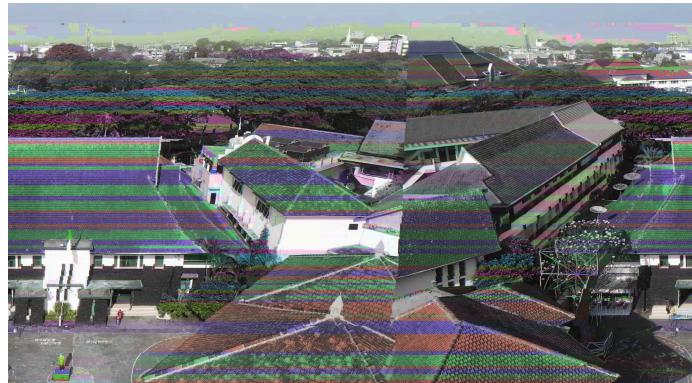
Output GIF digunakan agar dapat mendapatkan visualisasi progresif mengenai perkembangan Quad Tree setiap kedalaman detil yang dipakai. Implementasi menggunakan Java built-in GIF Encoder. Dengan menambahkan flag `'-g` atau `"--gif` maka program secara otomatis membuat GIF.

```
java.exe -jar Tucil2_13523045_13523052.jar -i .jpg1.jpg -m  
Variance -t 200 -b 80 -o out -gif
```

## BAB IV PERCOBAAN PROGRAM

### 4.1. Eksperimen 1

Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	vlcsnap-2022-12-06-19h15m00s581.png	Sumber	Dokumen Pribadi
Ukuran	5.29 MB (5,552,881 bytes)	Channel	96 dpi 24 bit
Lebar	2688	Tinggi	1512

Metode perhitungan error yang akan digunakan adalah metode Variance dengan threshold 200. Lalu ukuran blok minimum juga akan dibuat menjadi 8.

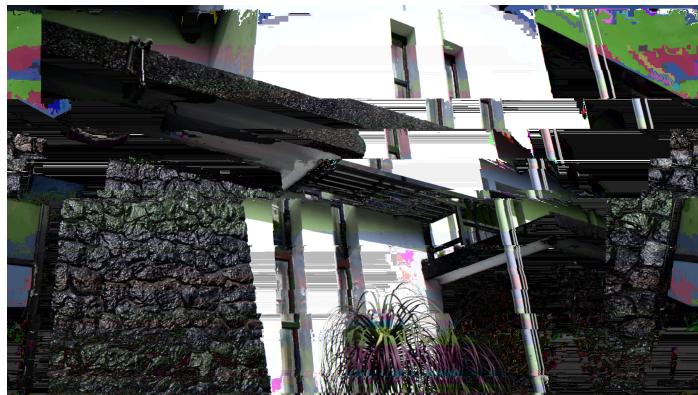




Gambar di atas merupakan animasi.  
Kunjungi link docs untuk melihatnya.

## 4.2. Eksperimen 2

Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	vlcsnap-2022-12-29-16h37m17s028.png	Sumber	Dokumen Pribadi
Ukuran	1.71 MB (1,798,656 bytes)	Channel	96 dpi 24 bit
Lebar	1920	Tinggi	1080

Metode perhitungan error yang akan digunakan adalah metode MeanAbsoluteDeviation dengan threshold 20. Lalu ukuran blok minimum juga akan dibuat menjadi 4.

```
D:\ABANG\ITB\Semester 4\Startegi Algoritma\tucil2\tucil2_13523045_13523052>java -jar build/libs/Tucil2_13523045_13523052.jar -i test/eksperimen2/vlcsnap-2022-12-29-16h37m17s028.png -m MeanAbsoluteDeviation -t 20 -b 4 -o test/eksperimen2/solutions -g
Precompute done in 207,07ms
Step 1 done in 24,43ms
Step 2 done in 6,84ms
Step 3 done in 24,96ms
Step 4 done in 9,73ms
Step 5 done in 4,87ms
Step 6 done in 5,59ms
Step 7 done in 15,73ms
Step 8 done in 59,51ms
Step 9 done in 18,97ms
Finalization done in 1,68ms
===== OUTPUT =====
Mengolah gambar dengan ukuran 1024x768 pixels (Untuk ukuran peta 1000x750 pixels)
Ukuran gambar sebelum: 1798656 bytes
Ukuran hasil kompresi: 158262 bytes
Percentase kompresi: 91,20%
Kedalaman pohon: 10
Jumlah simpul pohon: 26821
Gambar hasil kompresi terakhir: test/eksperimen2/solutions/output.png
Gambar hasil GIF: test/eksperimen2/solutions/output.gif
```

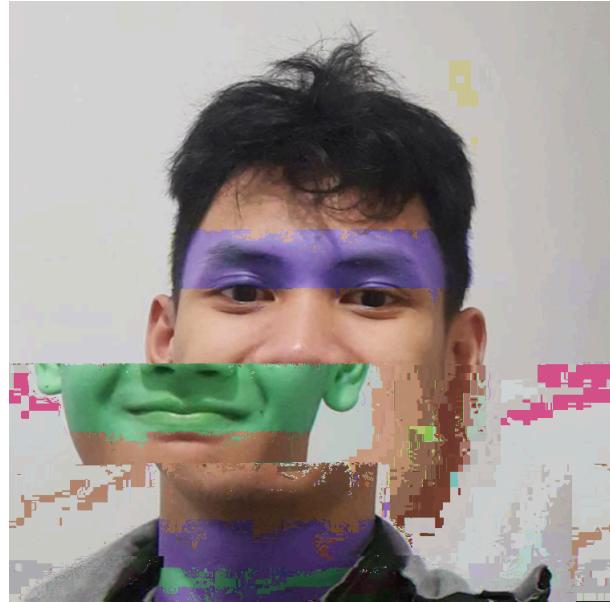


Gambar diatas merupakan animasi.

Kunjungi link docs untuk melihatnya.

### 4.3. Eksperimen 3

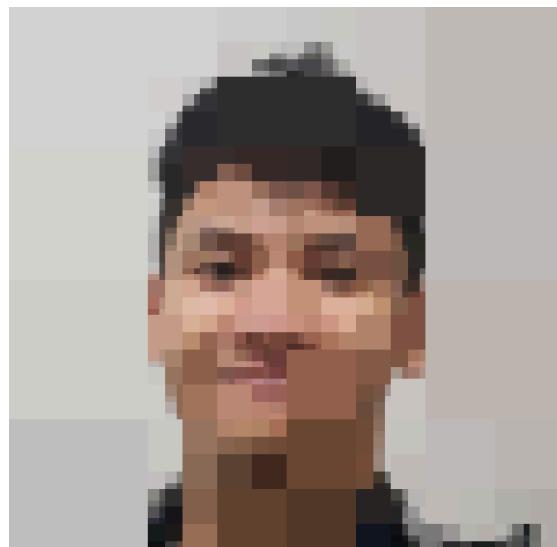
Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	nadhif.png	Sumber	Dokumen Pribadi
Ukuran	373 KB (381.979 bytes)	Channel	120 dpi 32 bit
Lebar	758	Tinggi	748

Metode perhitungan error yang akan digunakan adalah metode MeanAbsoluteDeviation dengan threshold 20. Lalu ukuran blok minimum juga akan dibuat menjadi 8.

```
D:\ABANG\ITB\Semester 4\Strategi Algoritma\tucil2\Tucil2_13523045_13523052>java -jar build/libs/Tucil2_13523045_13523052.jar -i test/eksperimen3/nadhif.png -m MeanAbsoluteDeviation -t 20 -b 8 -o test/eksperimen3/solutions -g
Precompute done in 92,70ms
Step 1 done in 13,81ms
Step 2 done in 5,58ms
Step 3 done in 12,38ms
Step 4 done in 1,52ms
Step 5 done in 0,78ms
Step 6 done in 0,48ms
Step 7 done in 1,99ms
Finalization done in 0,48ms
=====
OUTPUT
aktu eksekusi: 42,53ms (+waktu I/O 1370,70ms) (+waktu generate GIF 1102,86ms)
ukuran gambar sebelum: 381979 bytes
ukuran hasil kompresi: 26624 bytes
persentase kompresi: 93,03%
jmlahan pohon: 8
jumlah simpul pohon: 993
gambar hasil kompresi terakhir: test/eksperimen3/solutions/output.png
gambar hasil GIF: test/eksperimen3/solutions/output.gif
```



Gambar di atas merupakan animasi.  
Kunjungi link docs untuk melihatnya.

#### 4.4. Eksperimen 4

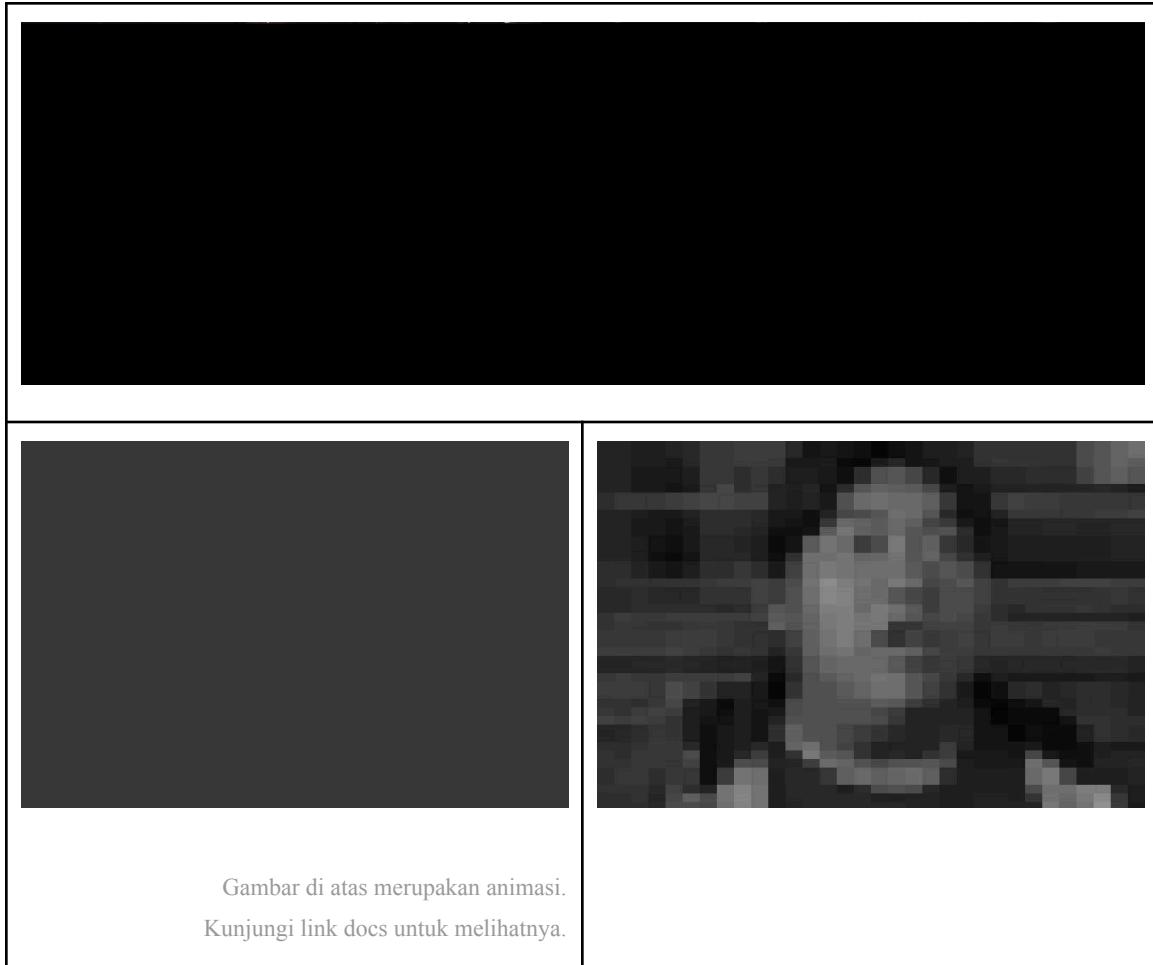
Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	orang.jpg	Sumber	Dokumen Pribadi
------	-----------	--------	-----------------

Ukuran	44,5 KB (45.652 bytes)	Channel	96 dpi 24 bit
Lebar	640	Tinggi	427

Metode perhitungan error yang akan digunakan adalah metode SSIM dengan threshold 0.7. Lalu ukuran blok minimum juga akan dibuat menjadi 10.



#### 4.5. Eksperimen 5

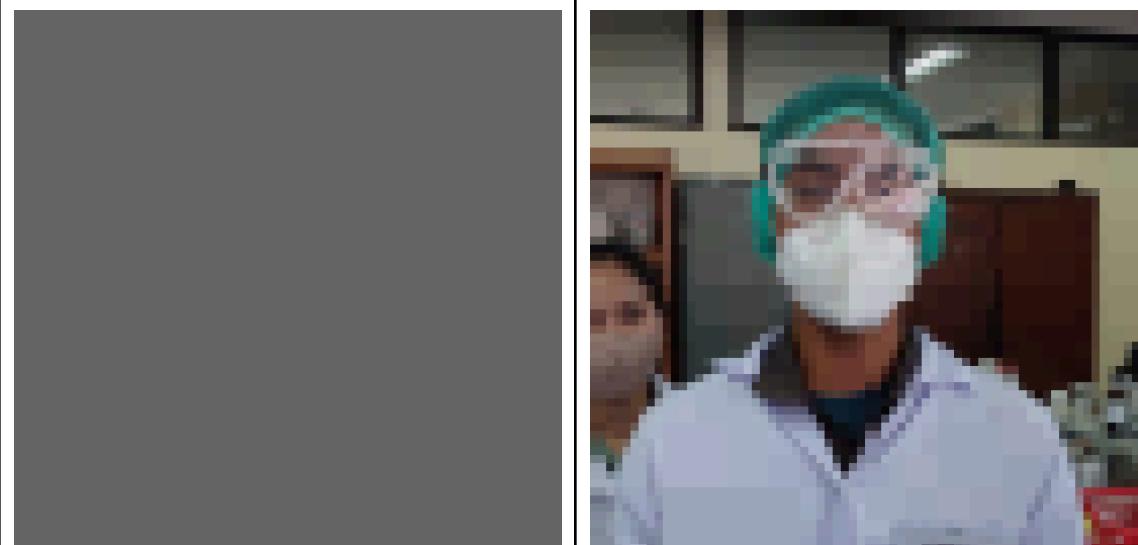
Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	aseton.png	Sumber	Dokumen Pribadi
Ukuran	335 KB (343.427 bytes)	Channel	120 dpi 32 bit
Lebar	542	Tinggi	532

Metode perhitungan error yang akan digunakan adalah metode MaxPixelDifference dengan threshold 30. Lalu ukuran blok minimum juga akan dibuat menjadi 8.

```
D:\ABANG\ITB\Semester 4\Startegi Algoritma\tucil2\tucil2_13523045_13523052>java -jar build/libs/Tucil2_13523045_13523052.jar -i test/eksperimen5/aseton.png -m MaxPixelDifference -t 30 -b 8 -o test/eksperimen5/solutions -g
Precompute done in 85,93ms
Step 1 done in 19,78ms
Step 2 done in 8,55ms
Step 3 done in 7,12ms
Step 4 done in 4,46ms
Step 5 done in 5,74ms
Step 6 done in 6,55ms
Step 7 done in 6,72ms
Finalization done in 14,44ms
[...-OUTPUT-]
Waktu eksekusi: 70,44ms (+waktu I/O 1534,12ms) (+waktu generate GIF 705,53ms)
Ukuran gambar sebelum: 343.427 bytes
Ukuran hasil kompresi: 2417 bytes
Persentase kompresi: 92,96%
Jumlah simpul pohon: 8
Ukuran hasil kompresi terakhir: test/eksperimen5/solutions/output.png
Ukuran hasil GIF: test/eksperimen5/solutions/output.gif
```



Gambar di atas merupakan animasi.  
Kunjungi link docs untuk melihatnya.

#### 4.6. Eksperimen 6

Pada eksperimen ini akan digunakan gambar sebagai berikut:

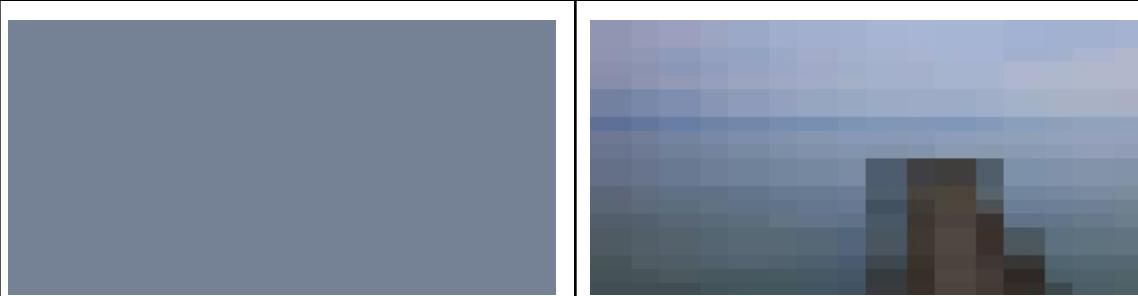
A photograph of a wooden pier extending into a calm sea under a cloudy sky.			
Nama	laut.jpeg	Sumber	<a href="https://jpeg.org/jpeg/">https://jpeg.org/jpeg/</a>
Ukuran	3,71 KB (3.800 bytes)	Channel	96 dpi 24 bit
Lebar	318	Tinggi	159

Metode perhitungan error yang akan digunakan adalah metode Entropy dengan threshold 3. Lalu ukuran blok minimum juga akan dibuat menjadi 8.

```

D:\VBTM&LITS\semester 4\Startegi Algoritma\luci12\luci12_13523045_13523052>java -jar build/lcks/luci12_13523045_13523052.jar -i test/eksperimen6/laut.jpg -m Entropy -t 3 -b 8 -o test/eksperimen6/solutions -g
Precompute done in 193,94ms
Step 1 done in 21,39ms
Step 2 done in 7,36ms
Step 3 done in 4,72ms
Step 4 done in 4,82ms
Step 5 done in 8,60ms
Finalization done in 0,56ms
----- OUTPUT -----
Waktu eksekusi: 47,46ms (+waktu I/O 208,51ms) (+waktu generate GIF 470,00ms)
Ukuran gambar sebelum: 3800 bytes
Ukuran hasil kompresi: 1865 bytes
Persentase kompresi: 50,92%
Verifikasi nohpe: 6
Jumlah simbol pohon: 541
hasil kompresi terakhir: test/eksperimen6/solutions/output.jpg
hasil GIF: test/eksperimen6/solutions/output.gif

```



Gambar di atas merupakan animasi.  
Kunjungi link docs untuk melihatnya.

## 4.7. Eksperimen 7

Pada eksperimen ini akan digunakan gambar sebagai berikut:



Nama	gunung.jpg	Sumber	<a href="https://www.pexels.com/search/4k/">https://www.pexels.com/search/4k/</a>

Ukuran	1,76 MB (1.847.928 bytes)	Channel	72 dpi 24 bit
Lebar	5472	Tinggi	3648

Metode perhitungan error yang akan digunakan adalah metode Variance dengan threshold 1. Lalu ukuran blok minimum juga akan dibuat menjadi 1.

```
D:\ABANG\ITB\Semester 4\Startegi Algoritma\tucil2\Tucil2_13523045_13523052>java -jar build/libs/Tucil2_13523045_13523052.jar -i test/eksperimen7/gunung.jpg -m Variance -t 1 -b 1 -o test/eksperimen7/solutions -g
Precompute done in 1373,46ms
Step 1 done in 194,89ms
Step 2 done in 129,81ms
Step 3 done in 75,42ms
Step 4 done in 49,41ms
Step 5 done in 25,84ms
Step 6 done in 25,84ms
Step 7 done in 57,55ms
Step 8 done in 122,63ms
Step 9 done in 197,05ms
Step 10 done in 699,40ms
Step 11 done in 1386,95ms
Step 12 done in 108,38ms
Initialization done in 2189,50ms
===== OUTPUT =====
aktu eksekusi: 8548,33ms (+waktu I/O 36466,03ms) (+waktu generate GIF 63547,38ms)
ukuran gambar sebelum: 1847928 bytes
ukuran hasil kompresi: 4818751 bytes
persentase kompresi: -160,77%
jumlah simpul pohon: 13
jumlah simpul pohon: 17824821
gambar hasil kompresi terakhir: test/eksperimen7/solutions/output.jpg
gambar hasil GIF: test/eksperimen7/solutions/output.gif
```



Gambar di atas merupakan animasi.  
Kunjungi link docs untuk melihatnya.

## BAB V ANALISIS

### 5.1. Analisis

Dari percobaan diatas didapatkan beberapa hasil yang dapat menurunkan ukuran gambar secara signifikan. Hal ini karena metode kompresi berbasis Quad Tree memanfaatkan struktur spasial dari gambar dan sifat homogenitas warna dalam blok-blok tertentu.

Quad Tree bekerja dengan cara membagi gambar menjadi blok-blok kuadran (empat bagian) secara rekursif dalam hal ini rekursif implisit. Setiap blok akan diuji apakah blok tersebut “cukup seragam” berdasarkan kriteria tertentu melalui pengukuran metode galat seperti variansi, *mean absolute deviation* (MAD), *max pixel difference* (MPD), Entropy, dan *structural similarity index measurement* (SSIM). Yang mana jika blok memiliki keseragaman yang dekat, maka blok tersebut tidak akan dipecah dan akan direpresentasikan oleh warna rata-rata.

Dengan begitu, blok tidak perlu menyimpan semua informasi semua pixel dan hanya cukup menyimpan satu simpul Quad Tree beserta warnanya. Sebaliknya, jika blok memiliki detil yang tinggi, maka blok akan dipecah lagi menjadi 4 sub-blok bagian.

Namun, terdapat kasus di mana metode kompresi ini memberikan ukuran file yang lebih besar dibanding ukuran aslinya. Pada percobaan terakhir menggunakan Variance dengan minimum blok 1 dan threshold 1 menghasilkan ukuran kompresi yang lebih besar dibanding ukuran aslinya. Hal ini terjadi karena threshold yang terlalu rendah dan minimum blok yang terlalu kecil. Dengan threshold yang renda (yakni 1), berarti blok akan dianggap seragam jika variansi warnanya kurang dari 1. Sedangkan dalam kehidupan nyata tetap mungkin ada perbedaan warna bahkan pada area “flat” sekalipun. Akibatnya semua blok dianggap tidak seragam. Selain itu, minimum blok size 1 mengizinkan pembagian blok hingga ukuran 1 x 1. Hal ini mengakibatkan jumlah blok yang akan dibuat sama dengan jumlah pixel pada gambar.

Dalam melakukan kompresi, diperlukan threshold dan minimum block size yang seimbang agar didapatkan gambar yang terkompresi dengan baik tetapi masih dapat mengatasi *trade off* kualitas dengan baik. Threshold yang terlalu rendah akan membuat algoritma terlalu sensitif terhadap perbedaan warna kecil, sehingga banyak blok yang tetap dipecah dan menyebabkan struktur pohon menjadi dalam dan kompleks. Hal ini dapat mengakibatkan ukuran file hasil kompresi menjadi lebih besar dari gambar aslinya. Sebaliknya, jika threshold terlalu tinggi, blok-blok dengan detail penting justru tidak akan dipecah, sehingga informasi visual dapat hilang secara signifikan dan kualitas gambar menurun. Demikian pula, pemilihan minimum block size yang terlalu kecil memungkinkan kompresi berjalan terlalu dalam hingga tingkat piksel, yang dapat mengurangi efisiensi dan memperbesar overhead metadata struktur pohon. Oleh karena itu, kombinasi nilai threshold yang moderat dan batas ukuran blok minimum yang tidak terlalu kecil

sangat krusial untuk menghasilkan kompresi yang efisien secara ruang, namun tetap menjaga struktur dan kualitas visual gambar.

## 5.2. Kompleksitas Algoritma

Pada program ini, Quad Tree menggunakan pendekatan Divide and Conquer dengan membuat 1 blok yang dapat dibagi menjadi 4 blok jika tidak memenuhi threshold tertentu. Misalkan saja memiliki jumlah blok sebesar lebar dan tinggi gambarnya dengan persamaan seperti berikut:

$$n = W \times H$$

Dengan  $n$  sebagai jumlah blok,  $W$  sebagai lebar, dan  $H$  sebagai tinggi. Didapatkan total blok yang memiliki kemungkinan untuk dilakukan split menjadi 4 blok bagian kecil. Karena blok dibuat dalam child tree dari Quad Tree, berarti blok dibentuk secara rekursif. Oleh karena itu dapat digunakan pendekatan persamaan teori master sebagai berikut:

$$T(n) = 4 \times T\left(\frac{n}{4}\right) + f(n)$$

Setiap kali melakukan split pada sebuah balok gambar berukuran  $w \times h$ , blok akan dibagi menjadi 4 subblok dalam hal ini 4 sub blok tersebut merupakan 4 sub masalah yang dipecah dari 1 masalah besar. Masing-masing sub masalah berukuran  $\frac{n}{4}$  dari total masalah yang ada (dalam hal ini luas total).  $f(n)$  dalam hal ini merupakan kompleksitas biaya diluar pemanggilan rekursif. Pada hal ini mencakup perhitungan rata-rata warna, *variance*, *entropy*, *SSIM*, dan sebagainya.

Dalam *worst case* nya, perhitungan metode *shouldSplit* pada perhitungan galat dapat diperlukan melakukan iterasi sebanyak  $w \times h$ . dalam hal ini menghitung waktu untuk satu blok dapat disederhanakan menjadi  $\theta(w \cdot h) = \theta(n)$ . Maka dari itu dapat diasumsikan  $f(n) = \theta(n)$ .

Dalam teorema master terdapat 3 kondisi dalam menentukan nilai *bigO*. Dalam kasus ini karena memiliki persamaan  $T(n) = 4 \times T\left(\frac{n}{4}\right) + \theta(n)$ , dapat ditentukan nilai a, b dan d. Dengan nilai  $a = 4$ ,  $b = 4$ , dan  $d = 1$ . Hal ini memenuhi kasus 2 dalam teori master yang menyebutkan  $O(n^d \log n)$  jika  $a = b^d$ . Maka kompleksitas waktu yang dimiliki oleh program ini adalah  $T(n) = O(n \log n)$ .

## 5.3. Kesimpulan

Pada Tugas Kecil kali ini, kelompok kami telah menerapkan algoritma Divide and Conquer dalam mengatasi persoalan untuk melakukan kompresi gambar menggunakan metode Quad Tree.

Dalam membuat program ini kami menggunakan bahasa Java dan melakukan beberapa eksperimen untuk melakukan kompresi gambar. Berdasarkan hasil uji coba kami, threshold dan minimum block size sangat memengaruhi kualitas dan ukuran gambar. Untuk mendapatkan kualitas gambar yang optimal dengan size sekecil mungkin diperlukan keseimbangan antara threshold dan minimum block size. Selain itu, kami melakukan analisis kompleksitas waktu pada program ini dan didapatkan kompleksitas waktu sebesar  $T(n) = O(n \log n)$ .

#### **5.4. Saran**

Tugas Kecil ini sudah cukup baik dan diberikan kelonggaran waktu yang sangat banyak juga dari asisten, Kami berterima kasih atas kebaikan asisten dalam memberikan kemudahan dalam mengerjakan tugas kecil ini. Saran bagi kelompok kami, akan lebih baik dan optimal jika tugas kecil ini dikerjakan dari jauh-jauh hari sehingga akan banyak pengoptimalan serta bonus yang dapat diimplementasikan. Namun, kami cukup berbangga karena dapat menyelesaikan tugas kecil ini dengan cukup baik. Saran kami bagi para asisten, mungkin diberi source yang lebih banyak lagi mengenai implementasi Quad Tree dan perhitungannya karena banyak sekali informasi yang perlu digali lebih dalam dari spek yang ada.

## BAB VI LAMPIRAN

### 6.1. Tautan

Github: [Tucil2\\_13523045\\_13523052](#)

Link Laporan:  [Tucil2\\_13523045\\_13523052](#)

### 6.2. Tabel Laporan

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	✓	
2. Program berhasil dijalankan	✓	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	✓	
4. Mengimplementasi seluruh metode perhitungan error wajib	✓	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan		
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8. Program dan laporan dibuat	✓	

(kelompok) sendiri		
--------------------	--	--