

# **IF2211 - Strategi Algoritma**

## **Laporan Tugas Kecil 2**

*Kompresi Gambar dengan Metode Quadtree*



**Disusun Oleh:**

Ranashahira Reztaputri                    13523007

Heleni Gratia M. Tampubolon            13523107

**Program Studi Teknik Informatika  
Sekolah Teknik Elektro dan Informatika  
Institut Teknologi Bandung  
2025**

## DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB 1.....</b>	<b>4</b>
<b>DESKRIPSI TUGAS.....</b>	<b>4</b>
<b>BAB 2.....</b>	<b>5</b>
<b>LANDASAN TEORI.....</b>	<b>5</b>
2.1 Algoritma Divide and Conquer.....	5
2.2. Quadtree.....	6
<b>BAB 3.....</b>	<b>9</b>
<b>APLIKASI STRATEGI DIVIDE AND CONQUER.....</b>	<b>9</b>
3.1. QuadTreeNode.java.....	9
3.1.1. Pseudocode.....	9
3.1.2. Algoritma.....	9
3.2. QuadTreeCompression.java.....	10
3.2.1. Fungsi Main.....	10
3.2.1.1. Pseudocode.....	10
3.2.1.2. Algoritma.....	12
3.2.2. Fungsi copyOf().....	14
3.2.2.1. Pseudocode.....	14
3.2.2.2. Algoritma.....	14
3.2.3. Fungsi buildQuadTreeBFS() dan buildQuadTreeBFSwithThreshold().....	14
3.2.3.1. Pseudocode.....	14
3.2.3.2. Algoritma.....	17
3.2.4. Procedure saveGif.....	18
3.2.4.1. Pseudocode.....	18
3.2.4.2. Algoritma.....	19
3.2.5. Procedure drawQuadTree.....	20
3.2.5.1. Pseudocode.....	20
3.2.5.2. Algoritma.....	20
3.2.6. Function findBestThreshold.....	21
3.2.6.1. Pseudocode.....	21
3.2.6.2. Algoritma.....	23
3.2.7. Function computeError.....	24
3.2.7.1. Pseudocode.....	24
3.2.7.2. Algoritma.....	25
3.2.8. Function computeVariance.....	26
3.2.7.1. Pseudocode.....	26
3.2.8.2. Algoritma.....	27
3.2.9. Function computeMAD.....	28
3.2.9.1. Pseudocode.....	28

3.2.9.2. Algoritma.....	29
3.2.10. Function computeMaxDiff.....	29
3.2.10.1. Pseudocode.....	29
3.2.10.2. Algoritma.....	30
3.2.11. Function averageColor.....	31
3.2.11.1. Pseudocode.....	31
3.2.11.2. Algoritma.....	32
3.2.12. Function computeEntropy.....	32
3.2.12.1. Pseudocode.....	32
3.2.12.2. Algoritma.....	33
3.2.13. Function computeSSIM.....	34
3.2.13.1. Pseudocode.....	34
3.2.13.2. Algoritma.....	35
3.2.14. Procedure getSerializedSize.....	36
3.2.14.1. Pseudocode.....	36
3.2.14.2. Algoritma.....	36
3.2.15. Procedure printTableHeader, printRow, and printTableFooter.....	37
3.2.15.1. Pseudocode.....	37
3.2.15.2. Algoritma.....	37
<b>BAB 4.....</b>	<b>39</b>
<b>IMPLEMENTASI DAN PENGUJIAN.....</b>	<b>39</b>
4.1. Implementasi.....	39
4.1.1. QuadTreeCompression.java.....	39
4.1.2. GifSequenceWriter.java.....	50
4.2. Struktur Data.....	50
4.2.1 Struktur Data Repository.....	50
4.3. Pengujian.....	52
4.3.1. Pengujian 1.....	52
4.3.2. Pengujian 2.....	53
4.3.4. Pengujian 4.....	55
4.3.5. Pengujian 5.....	56
4.3.6. Pengujian 6.....	57
4.3.7. Pengujian 7.....	58
4.4. Analisis Hasil Pengujian.....	59
<b>BAB 5.....</b>	<b>63</b>
<b>KESIMPULAN DAN SARAN.....</b>	<b>63</b>
5.1. Kesimpulan.....	63
5.2. Saran.....	63
<b>LAMPIRAN.....</b>	<b>65</b>
Tautan Repository GitHub.....	65
<b>DAFTAR PUSTAKA.....</b>	<b>66</b>

# BAB 1

## DESKRIPSI TUGAS



Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan. Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

## **BAB 2**

### **LANDASAN TEORI**

#### **2.1 Algoritma Divide and Conquer**

Algoritma *Divide and Conquer* merupakan salah satu paradigma pemrograman yang efektif dan banyak digunakan untuk menyelesaikan permasalahan komputasi yang kompleks. Paradigma ini bekerja dengan membagi suatu persoalan besar menjadi beberapa sub-persoalan yang lebih kecil namun serupa dengan persoalan awal. Setiap sub-persoalan kemudian diselesaikan secara mandiri, dan hasil dari penyelesaian sub-persoalan tersebut akan digabungkan untuk memperoleh solusi dari persoalan utama.

Metode ini terdiri dari tiga tahap utama:

##### **1. Divide (Pemisahan)**

Pada tahap ini, persoalan utama berukuran besar dibagi menjadi satu atau lebih sub-persoalan yang memiliki struktur serupa namun dengan ukuran yang lebih kecil. Pembagian ini dilakukan hingga mencapai persoalan terkecil yang dapat diselesaikan secara langsung.

##### **2. Conquer (Penyelesaian)**

Setiap sub-persoalan yang dihasilkan kemudian diselesaikan. Jika sub-persoalan masih cukup besar, proses pemisahan dilakukan kembali secara rekursif. Jika sudah cukup kecil (biasanya disebut *base case*), maka dapat diselesaikan secara langsung tanpa rekursi.

##### **3. Combine (Penggabungan)**

Setelah semua sub-persoalan diselesaikan, hasilnya digabungkan untuk membentuk solusi dari persoalan semula.

Objek persoalan yang biasanya cocok untuk pendekatan ini adalah data masukan berukuran nnn, seperti tabel, matriks, eksponen, dan polinom. Setiap sub-persoalan memiliki karakteristik yang serupa dengan persoalan utama, hanya dalam skala yang lebih kecil. Oleh karena itu, metode *Divide and Conquer* secara alami diimplementasikan menggunakan skema rekursif.

Beberapa contoh algoritma yang menggunakan paradigma ini antara lain:

##### **1. Merge Sort dan Quick Sort untuk pengurutan data**

2. Binary Search untuk pencarian dalam array terurut
3. Mencari sepasang titik terdekat (*closest pair problem*)
4. Kompresi gambar menggunakan QuadTree

Keunggulan utama dari algoritma *Divide and Conquer* terletak pada efisiensinya dalam menyelesaikan persoalan yang kompleks, terutama dalam hal pengurangan kompleksitas waktu. Selain itu, pendekatan ini juga memungkinkan pemrosesan paralel pada sub-persoalan, yang sangat berguna dalam implementasi berbasis arsitektur komputasi paralel atau terdistribusi.

## 2.2. Quadtree

Quadtree adalah struktur data hierarkis berbasis pohon yang digunakan untuk merepresentasikan data dua dimensi dengan cara membagi ruang menjadi empat bagian (kuadran) pada setiap tingkat. Struktur ini banyak digunakan dalam berbagai bidang seperti grafika komputer, pemrosesan citra digital, pengindeksan spasial, dan kompresi data. Dalam konteks kompresi gambar, Quadtree dimanfaatkan untuk membagi citra menjadi blok-blok yang lebih kecil berdasarkan keseragaman warna atau intensitas piksel. Tujuannya adalah untuk menyimpan hanya informasi yang esensial, terutama pada bagian gambar yang tidak memiliki banyak variasi, sehingga data dapat dikompresi secara efisien.

Proses kompresi gambar dengan Quadtree terdiri dari beberapa tahap berikut:

### 1. *Divide (Pembagian)*

Gambar awal dianggap sebagai satu blok besar, lalu dilakukan pembagian menjadi empat bagian (kuadran) secara rekursif.

### 2. *Cek Keseragaman Blok*

Tiap blok yang terbentuk diperiksa apakah nilai piksel di dalamnya seragam. Pemeriksaan ini biasanya dilakukan dengan menghitung nilai rata-rata warna (dalam format RGB) dan mengevaluasi variasinya terhadap ambang batas tertentu (threshold). Jika perbedaan antar piksel masih dalam batas threshold, blok dianggap seragam.

### 3. *Conquer dan Recursion*

Jika blok tidak seragam dan ukurannya masih di atas batas minimum tertentu (misalnya minimum Block Size), maka blok tersebut dibagi lagi menjadi empat bagian dan proses diulangi. Jika ukuran blok sudah mencapai nilai minimum, maka pembagian dihentikan meskipun blok masih belum seragam.

#### 4. *Combine* dan Penyimpanan

Blok-blok yang dinyatakan seragam tidak akan dibagi lagi dan direpresentasikan sebagai simpul daun (leaf node). Simpul-simpul ini menyimpan data seperti posisi blok (x, y), ukuran (width, height), dan nilai warna rata-rata blok tersebut. Sedangkan simpul yang masih memiliki anak disebut simpul internal, dan menunjukkan area yang masih membutuhkan pembagian lebih lanjut.

Secara teknis, setiap simpul pada struktur Quadtree memiliki maksimal empat anak. Struktur ini dapat direpresentasikan secara rekursif, dan setiap simpul mengandung informasi sebagai berikut:

- Posisi blok (x, y)
- Ukuran blok (lebar dan tinggi)
- Nilai rata-rata warna atau intensitas piksel
- Daftar anak simpul (jika ada)

Struktur pohon ini tidak hanya mempermudah navigasi data spasial, tetapi juga menghilangkan redundansi pada area gambar yang seragam, sehingga sangat cocok untuk digunakan dalam algoritma kompresi *lossy* — yakni kompresi yang memungkinkan kehilangan sebagian data tetapi tetap mempertahankan kualitas visual yang baik.

Penggunaan struktur Quadtree dalam kompresi gambar menawarkan berbagai keunggulan yang membuatnya menjadi metode yang efisien dan fleksibel. Salah satu keunggulan utamanya adalah efisiensi penyimpanan. Pada gambar yang memiliki area dengan warna atau intensitas yang seragam, Quadtree hanya menyimpan satu simpul yang merepresentasikan seluruh area tersebut, tanpa perlu mencatat informasi setiap piksel di dalamnya. Hal ini secara signifikan mengurangi jumlah data yang harus disimpan. Selain itu, Quadtree secara alami mendukung representasi multiresolusi, karena pembagian gambar dilakukan secara hierarkis dari skala besar ke skala yang lebih kecil. Ini memungkinkan gambar direpresentasikan dengan tingkat detail yang bervariasi sesuai kebutuhan.

Keunggulan lainnya adalah kemampuan untuk mengontrol ukuran file hasil kompresi dan kualitas visualnya melalui penggunaan parameter seperti threshold dan minBlockSize. Dengan mengatur threshold, pengguna dapat menentukan seberapa besar variasi warna yang masih dianggap seragam, sedangkan minimum BlockSize mencegah pembagian blok menjadi terlalu

kecil, menjaga keseimbangan antara efisiensi kompresi dan kualitas gambar. Selain itu, struktur pohon yang dimiliki Quadtree mempermudah proses rendering atau pencarian spasial dalam gambar, karena akses terhadap area tertentu dapat dilakukan lebih cepat dan terorganisir. Dengan semua keunggulan tersebut, Quadtree menjadi pilihan yang ideal untuk kompresi gambar terutama dalam konteks aplikasi yang membutuhkan efisiensi tinggi tanpa mengorbankan informasi visual yang penting.

## BAB 3

### APLIKASI STRATEGI DIVIDE AND CONQUER

#### 3.1. QuadTreeNode.java

##### 3.1.1. Pseudocode

```
class QuadTreeNode{
    input x, y, width, height : integer;
    input color : integer;
    input isLeaf : boolean;
    output node : QuadTreeNode)

    { Membuat simpul Quadtree baru }

Deklarasi:
    i : integer

Algoritma:
    node.x ← x
    node.y ← y
    node.width ← width
    node.height ← height
    node.color ← color
    node.isLeaf ← isLeaf

    if isLeaf = true then
        node.children ← null
    else
        for i ← 1 to 4 do
            node.children[i] ← null
        endfor
    endif
```

##### 3.1.2. Algoritma

Kelas QuadTreeNode merupakan representasi dari sebuah simpul (node) dalam struktur data Quadtree, yang digunakan untuk menyimpan informasi tentang bagian-bagian gambar dalam proses kompresi atau representasi spasial. Kelas ini mengimplementasikan antarmuka Serializable, yang berarti objek dari kelas ini dapat disimpan ke file atau ditransmisikan melalui jaringan dalam bentuk data biner (serialization).

Parameter	Type	Keterangan
-----------	------	------------

x, y	integer	Koordinat posisi pojok kiri atas dari blok gambar yang diwakili oleh simpul.
width, height	integer	Ukuran (lebar dan tinggi) dari blok gambar.
color	integer	Nilai warna rata-rata (biasanya dalam format RGB) yang dikodekan sebagai satu integer) dari seluruh piksel dalam blok ini. Digunakan saat simpul adalah lead
isLeaf	boolean	Menunjukkan apakah simpul ini adalah simpul daun atau bukan. Jika true, maka simpul tidak memiliki anak dan mewakili area seragam. Jika false, maka simpul ini memiliki anak-anak yang merepresentasikan pembagian lebih lanjut dari blok.
children	QuadTreeNode[]	Array berisi maksimal empat simpul anak (child) yang mewakili subdivisi dari blok gambar ini. Jika simpul adalah daun, maka array ini akan bernilai null.

## 3.2. QuadTreeCompression.java

### 3.2.1. Fungsi Main

#### 3.2.1.1. Pseudocode

```
procedure Main()
{ Melakukan kompresi gambar menggunakan metode QuadTree }
```

**Deklarasi:**

img : Gambar  
 imagePath : string  
 originalSize, executionTime, compressedSize : long  
 targetCompression, threshold : real  
 outputName, gifName, outputFolder, outputPath, gifPath : string  
 minBlockSize, method : integer  
 isMinBlock : boolean  
 root : QuadTreeNode

**Algoritma:**

1. LOOP Validasi Gambar  
 printTableHeader()

```

While (gambar tidak valid)
    Output (Masukkan nama gambar (.jpg/.png): )
    Input (imageName)
    imagePath ← "../test/source/" + imageName
    if (file tidak ditemukan atau rusak)
        output(pesan kesalahan)
    else (gambar valid)
        printRow(imageName)
    endwhile
until gambar valid

```

2. Hitung ukuran file asli:

```
originalSize ← ukuran file gambar
```

3. Input metode error dan pengaturan kompresi:

```
Output (Pilih metode error (1–5): )
```

```
Input(method)
```

```
printRow(method)
```

```
Output(Masukkan target persentase kompresi (0 = nonaktif): )
```

```
Input(targetCompression)
```

```
printRow(targetCompression)
```

```
if targetCompression = 0:
```

```
    isMinBlock ← true
```

```
    input (threshold dan minBlockSize)
```

```
    printRow(threshold dan minBloksSize)
```

```
else:
```

```
    isMinBlock ← false
```

```
    threshold ← cariThresholdTerbaik(img, originalSize)
```

4. Input nama file hasil:

```
output (Masukkan nama gambar hasil: )
```

```
input(outputName)
```

```
printRow(outputName)
```

```
output (Masukkan nama file GIF hasil: )
```

```
input (gifName)
```

```
printRow(gifName)
```

```
print TableFooter()
```

5. Inisialisasi gambar awal:

```
buat currentFrame berukuran gambar asli
```

```
warnai dengan warna rata-rata
```

6. Bangun pohon QuadTree:

```
root ← buildQuadTreeBFS(img)
```

7. Simpan animasi ke GIF:

```
saveGif(gifPath, frames, delay=500ms)
```

8. Gambar ulang hasil kompresi dari QuadTree:

```

drawQuadTree(compressedImage, root)
simpan ke outputPath

9. Hitung dan tampilkan hasil kompresi:
compressedSize ← ukuran serialisasi pohon
compressionPercentage ← 100 - (compressedSize / originalSize * 100)

printTableHeader()
printRow(waktu eksekusi, ukuran awal, ukuran akhir, persentase kompresi)
printRow(kedalaman maksimal pohon dan total simpul)
output(lokasi penyimpanan hasil gambar dan gif)

```

### 3.2.1.2. Algoritma

Program ini bertujuan untuk melakukan kompresi gambar dengan pendekatan struktur data QuadTree, yang membagi gambar menjadi blok-blok kuadrat lebih kecil berdasarkan nilai ambang batas error tertentu. Proses dimulai dengan meminta pengguna untuk memasukkan nama gambar yang akan dikompresi. Validasi dilakukan agar gambar benar-benar ada dan bisa dibaca. Setelah itu, ukuran asli gambar dihitung, lalu pengguna diminta memilih metode pengukuran error yang digunakan dalam pemisahan blok (seperti Variansi, MAD, Max Pixel Difference, Entropy, atau SSIM).

Jika pengguna menentukan target persentase kompresi, maka sistem secara otomatis mencari nilai threshold yang sesuai untuk mencapai target tersebut. Jika tidak, pengguna dapat memasukkan threshold dan ukuran blok minimum secara manual. Setelah semua parameter ditentukan, nama file hasil gambar terkompresi dan GIF animasi ditentukan, dan folder penyimpanan hasil dibuat jika belum tersedia.

Selanjutnya, gambar awal diinisialisasi dengan warna rata-rata, dan pohon QuadTree dibangun menggunakan kombinasi algoritma *Divide and Conquer* dan traversal *Breadth-First Search* (BFS). Proses ini merekam perubahan setiap pembagian blok sebagai frame untuk GIF animasi. Setelah pohon selesai dibentuk, gambar hasil kompresi digambar ulang berdasarkan struktur pohon dan disimpan. Terakhir, program menghitung ukuran hasil kompresi, membandingkannya dengan ukuran asli, dan menampilkan informasi statistik seperti waktu eksekusi, tingkat kompresi, kedalaman pohon, dan total simpul.

Parameter	Type	Keterangan
img	BufferedImage	Objek gambar yang dibaca dari file input.
imagePath	width, height	Path lengkap ke gambar input.
originalSize	long	Ukuran file gambar sebelum dikompresi (dalam byte).
executionTime	long	Lama waktu proses kompresi (dalam nanodetik).
compressedSize	long	Ukuran hasil kompresi dalam bentuk representasi serialisasi pohon.
targetCompression	double	Persentase kompresi yang ditargetkan oleh pengguna.
threshold	double	Nilai ambang batas error pemisahan blok dalam QuadTree.
outputName	String	Nama file untuk menyimpan gambar hasil.
gifName	String	Nama file GIF untuk animasi hasil proses pembagian blok.
outputFolder	String	Lokasi folder tempat hasil disimpan.
outputPath	String	Path lengkap untuk menyimpan gambar hasil kompresi.
gifPath	String	Path lengkap untuk menyimpan GIF hasil animasi.
minBlockSize	int	Ukuran terkecil dari blok dalam QuadTree jika tidak otomatis.
method	int	Pilihan metode error dari 1 sampai 5.
isMinBlock	boolean	Penanda apakah menggunakan threshold manual atau otomatis.
root	QuadTreeNode	Simpul akar dari pohon QuadTree hasil kompresi.

### 3.2.2. Fungsi copyOf()

#### 3.2.2.1. Pseudocode

```
function copyOf(input img : BufferedImage)
{ Menyalin gambar IMG ke BufferedImage baru dengan ukuran dan tipe yang sama
  Masukan : img berupa BufferedImage
  Luaran : salinan BufferedImage dari img }
```

#### Deklarasi

```
copy : BufferedImage
g      : Graphics2D
```

#### Algoritma

```
copy ← BufferedImage(lebar, tinggi, tipe ARGB)
g ← objek grafis dari copy
gambar img ke dalam copy pada posisi (0, 0) menggunakan g
buang sumber daya g
→ copy
```

### 3.2.2.2. Algoritma

Fungsi copyOf() menyalin BufferedImage img untuk dijadikan frame dalam animasi gif. img disalin supaya saat ditambahkan ke frame, yang ditambahkan adalah hasil salinannya bukan referensnya sehingga tiap frame akan berbeda beda sesuai state-nya saat itu.

Parameter	Type	Keterangan
img	BufferedImage	Gambar yang diinput oleh pengguna

### 3.2.3. Fungsi buildQuadTreeBFS() dan buildQuadTreeBFWithThreshold()

#### 3.2.3.1. Pseudocode

```
function buildQuadTreeBFS(input img : BufferedImage)
{ Membangun quadtree dari gambar img menggunakan algoritma BFS
  Masukan : img berupa BufferedImage
  Luaran : node akar quadtree }
```

#### Deklarasi

```
queue : queue of QuadTreeNode
root : QuadTreeNode
g : Graphics2D
node : QuadTreeNode
children : array [0..3] of QuadTreeNode
newChildren, currentLeaves : list of QuadTreeNode
size, i : integer
```

```

err : real
halfWidth, halfHeight : integer
maxDepth : integer

Algoritma
    root ← QuadTreeNode(0, 0, lebar, tinggi, warna rata-rata, false)
    queue ← queue berisi root
    currentFrame ← BufferedImage baru berukuran img.width × img.height bertipe
ARGB
    frames.clear()

    g ← currentFrame.createGraphics()
    g.setColor(new Color(root.color))
    g.fillRect(0, 0, img.width, img.height)
    g.dispose()
    frames.add(copyOf(currentFrame))

    currentLeaves ← list kosong

    while queue tidak kosong do
        size ← queue.size()
        newChildren ← list kosong

        for i ← 1 to size do
            node ← queue.poll()
            err ← computeError(img, node.x, node.y, node.width, node.height)
            halfWidth ← node.width ÷ 2
            halfHeight ← node.height ÷ 2

            if (isMinBlock dan (halfWidth × halfHeight < minBlockSize atau err <
threshold))
                atau (bukan isMinBlock dan err < threshold) then
                    node.isLeaf ← true
                    totalNodes ← totalNodes + 1
                    currentLeaves.add(node)
                else
                    Buat 4 children dari node dengan averageColor masing-masing
                    node.children ← children
                    Tambahkan semua children ke newChildren
                endfor

            if newChildren tidak kosong then
                g ← currentFrame.createGraphics()
                for setiap leaf dalam currentLeaves do
                    g.setColor(new Color(leaf.color))
                    g.fillRect(leaf.x, leaf.y, leaf.width, leaf.height)
                endfor
                for setiap child dalam newChildren do
                    maxDepth ← maksimum dari (maxDepth, child.depth)
                    g.setColor(new Color(child.color))

```

```

g.fillRect(child.x, child.y, child.width, child.height)
endfor
g.dispose()
frames.add(copyOf(currentFrame))
queue.addAll(newChildren)
endif
endwhile

→ root

```

**function** buildQuadTreeBFSwithThreshold(**input** img : BufferedImage, **input** thresholdVal : **real**)  
{ Membangun quadtree dari gambar img menggunakan BFS dan ambang batas error  
thresholdVal  
    Masukan : img berupa BufferedImage, thresholdVal sebagai batas error  
    Luaran : node akar quadtree }

#### Deklarasi

```

queue : queue of QuadTreeNode
root : QuadTreeNode
g : Graphics2D
node : QuadTreeNode
children : array [0..3] of QuadTreeNode
newChildren, currentLeaves : list of QuadTreeNode
size, i : integer
err : real
halfWidth, halfHeight : integer

```

#### Algoritma

```

root ← QuadTreeNode(0, 0, img.width, img.height, averageColor(img, 0, 0,
img.width, img.height), false)
queue ← queue berisi root
currentFrame ← BufferedImage baru berukuran img.width × img.height bertipe
ARGB
frames.clear()

g ← currentFrame.createGraphics()
g.setColor(new Color(root.color))
g.fillRect(0, 0, img.width, img.height)
g.dispose()
frames.add(copyOf(currentFrame))

currentLeaves ← list kosong

while queue tidak kosong do
    size ← queue.size()
    newChildren ← list kosong

```

```

for i ← 1 to size do
    node ← queue.poll()
    err ← computeError(img, node.x, node.y, node.width, node.height)

    if err < thresholdVal then
        node.isLeaf ← true
        totalNodes ← totalNodes + 1
        currentLeaves.add(node)
    else
        halfWidth ← node.width ÷ 2
        halfHeight ← node.height ÷ 2

        Buat 4 children dari node dengan ukuran dan averageColor masing-masing
        children[0] ← QuadTreeNode(node.x, node.y, halfWidth, halfHeight, ...)
        children[1] ← QuadTreeNode(node.x + halfWidth, node.y, node.width -
        halfWidth, halfHeight, ...)
        children[2] ← QuadTreeNode(node.x, node.y + halfHeight, halfWidth,
        node.height - halfHeight, ...)
        children[3] ← QuadTreeNode(node.x + halfWidth, node.y + halfHeight,
        node.width - halfWidth, node.height - halfHeight, ...)

        node.children ← children
        Tambahkan semua children ke newChildren
    endfor

    If newChildren ≠ null then
        Tambahkan elemen newChildren ke dalam queue
    endwhile
    → root

```

### 3.2.3.2. Algoritma

Fungsi buildQuadTreeBFS() membuat quadtree dengan memanfaatkan algoritma divide and conquer dan BFS (breadth first search). Algoritma secara keseluruhan menggunakan divide and conquer untuk membagi gambar (img) menjadi bagian yang lebih kecil dengan bentuk yang serupa (QuadtreeNode). Proses pembagian blok menjadi empat subblok berhenti ketika error blok berada dibawah threshold atau ketika ukuran blok setelah dibagi empat menjadi kurang dari minimum block size. Fungsi ini memiliki komponen-komponen dari divide and conquer, yaitu divide, conquer, dan combine. Proses divide pada fungsi ini adalah membagi blok gambar menjadi empat subblok. Proses conquer pada fungsi ini adalah mencari nilai error dan mencari warna rata-rata pada sunblock. Terakhir, proses combine pada fungsi ini adalah menyatukan blok yang sudah dibagi menjadi subblok children menjadi node yang memiliki atribut children. Selain divide and conquer, fungsi ini juga memanfaatkan BFS untuk membuat frame

ketika memproses output berupa gif. BFS dibutuhkan karena tiap frame berisi proses per tahap pembagian blok yang berada pada kedalaman yang sama, lalu frame berikutnya akan berisi tahap pembagian yang berada pada kedalaman berikutnya.

Hal yang sama terdapat pada fungsi buildQuadTreeWithThreshold(). Namun, terdapat beberapa modifikasi seperti tambahan parameter threshold dan penghilangan syarat minimum block size. Fungsi ini digunakan untuk pencarian nilai threshold yang mampu menghasilkan nilai persentase kompresi yang mendekati target.

Parameter	Type	Keterangan
img	BufferedImage	Gambar yang di input oleh pengguna
thresholdVal	double	Nilai threshold

### 3.2.4. Procedure saveGif

#### 3.2.4.1. Pseudocode

```
Procedure saveGif (path: string, frames: list of BufferedImage, delayMs: integer)
{ Menyimpan list gambar sebagai animasi GIF ke path yang ditentukan }
```

**Deklarasi:**

```
output : ImageOutputStream
writer : GifSequenceWriter
frame : BufferedImage
i : integer
```

**Algoritma:**

1. Buka stream untuk file GIF tujuan:  
output ← buka ImageOutputStream dari file di lokasi 'path'
2. Inisialisasi writer GIF:  
writer ← GifSequenceWriter(output, tipe gambar ARGB, delayMs, loop = true)
3. Tulis setiap frame ke dalam GIF:  
i ← 1  
**for** setiap frame dalam frames:  
    writer.writeToSequence(frame)  
    (opsiional: simpan juga frame sebagai file PNG)  
    i ← i + 1
4. Tutup writer dan output stream:  
writer.close()

```
output.close()
```

### 3.2.4.2. Algoritma

Procedure saveGif bertugas menyimpan sekumpulan gambar (frame) ke dalam satu file animasi GIF. Algoritma dimulai dengan membuka stream keluaran (ImageOutputStream) menuju path file tujuan, yang ditentukan oleh parameter path. Setelah stream dibuka, dibuatlah objek GifSequenceWriter, yaitu utilitas khusus yang memungkinkan penulisan beberapa gambar ke dalam format GIF secara berurutan. Writer ini dikonfigurasi dengan tipe gambar ARGB agar mendukung transparansi, dan diberi parameter delay sesuai nilai delayMs untuk menentukan durasi tiap frame. Selanjutnya, algoritma melakukan iterasi terhadap seluruh elemen dalam daftar frames, dan setiap BufferedImage dituliskan ke dalam urutan GIF melalui metode writeToSequence. Setelah seluruh frame berhasil dimasukkan ke file, proses diakhiri dengan menutup writer dan output stream untuk memastikan file GIF tersimpan dengan benar.

Parameter	Type	Keterangan
path	String	Jalur (path) file tujuan tempat menyimpan file GIF.
frames	List<BufferedImage>	Daftar gambar (frame) yang akan ditulis ke dalam animasi GIF.
delayMs	integer	Waktu jeda antar frame dalam milidetik (ms).
output	ImageOutputStream	Objek stream untuk menulis data ke file GIF.
writer	GifSequenceWriter	Kelas utilitas untuk menulis frame-frame ke dalam GIF secara berurutan.
frame	BufferedImage (per elemen list)	Gambar bitmap berwarna/transparan yang akan dijadikan satu file GIF.
i	integer	Variabel penghitung iterasi.

### 3.2.5. Procedure drawQuadTree

#### 3.2.5.1. Pseudocode

```
Procedure drawQuadTree(image, node)
{ Menggambar ulang gambar berdasarkan struktur QuadTree }
```

**Input:**

image : BufferedImage  
node : QuadTreeNode

**Algoritma:**

```
if node adalah daun (isLeaf = true) then
    buat objek grafik dari image
    atur warna berdasarkan node.color
    gambar persegi panjang pada posisi (node.x, node.y)
        dengan ukuran (node.width, node.height)
    tutup objek grafik
else
    untuk setiap anak dalam node.children do
        if child ≠ null then
            drawQuadTree(image, child)
```

#### 3.2.5.2. Algoritma

Algoritma drawQuadTree merupakan implementasi dari pendekatan *Divide and Conquer*, yang digunakan untuk menggambar ulang gambar hasil kompresi berdasarkan struktur QuadTree. Prinsip *Divide and Conquer* dalam algoritma ini diterapkan sebagai berikut:

##### 1. Divide (Pecah Masalah):

Jika node bukan daun (isLeaf = false), maka area gambar yang diwakilinya akan dibagi menjadi empat bagian (kuadran). Pembagian ini dilakukan sebelumnya saat pembuatan pohon QuadTree, dan di sini direkonstruksi dengan cara memproses keempat anak node tersebut satu per satu.

##### 2. Conquer (Selesaikan Submasalah):

Fungsi ini dipanggil secara rekursif untuk setiap anak node, sehingga setiap bagian gambar akan digambar ulang dari daun hingga akar.

##### 3. Combine (Gabungkan Solusi Submasalah):

Karena penggambaran dilakukan langsung ke dalam objek BufferedImage, hasil

setiap pemrosesan bagian (submasalah) langsung digabung dalam gambar akhir tanpa memerlukan proses penggabungan terpisah.\

Ketika algoritma menemukan node yang merupakan **daun**, proses pembagian berhenti dan gambar untuk area tersebut langsung diwarnai berdasarkan warna rata-rata yang disimpan dalam node.

Parameter	Type	Keterangan
img	Buffered Image	Gambar kosong yang akan diisi ulang berdasarkan struktur QuadTree.
node	QuadTreeNode	Node dalam pohon QuadTree yang menyimpan informasi posisi, ukuran, warna, dan anak-anaknya.
g	Graphics2D	Objek grafis untuk menggambar ke dalam BufferedImage. Dibuat dari img.
node.x/node.y	integer	Koordinat sudut kiri atas dari area yang diwakili node dalam gambar.
node.width/ node.height	integer	Lebar dan tinggi area yang diwakili node.
frame	BufferedImage (per elemen list)	Gambar bitmap berwarna/transparan yang akan dijadikan satu file GIF.
node.children	QuadTreeNode[] (array)	Menyimpan keempat anak node (jika node bukan daun).

### 3.2.6. Function findBestThreshold

#### 3.2.6.1. Pseudocode

```
Function findBestThreshold(img: Gambar, originalSize: Long) → Real
```

**Deklarasi:**

```
low, high, bestThreshold, mid: Real
bestCompressionDiff, diff, achievedCompression: Real
compressed: Integer
root: QuadTreeNode
iteration, count: Integer
highest: Real
```

**Inisialisasi:**

```
low ← 0
```

```

high ← 10
bestCompressionDiff ← tak hingga
achievedCompression ← 0
iteration ← 0

1. Perluas batas atas jika hasil kompresi masih < target
While (true):
    root ← buildQuadTreeBFSwithThreshold(img, high)
    compressed ← getSerializedSize(root)
    achievedCompression ← 1.0-(compressed/originalSize)

    If achievedCompression ≥ targetCompression:
        break
    high ← high + 500

    bestThreshold ← high
    highest ← high
    diff ← |achievedCompression - targetCompression|

2. Binary search hingga selisih cukup kecil atau iterasi maksimal
While iteration ≤ 5 AND diff > 0.001:
    count ← 0

    While diff > 0.001 AND count ≤ 5:
        mid ← (low + high) / 2
        root ← buildQuadTreeBFSwithThreshold(img, mid)
        compressed ← getSerializedSize(root)
        achievedCompression←1.0-(compressed/originalSize)
        diff ← |achievedCompression - targetCompression|

    If achievedCompression ≥0 AND diff< bestCompressionDiff:
        bestCompressionDiff ← diff
        bestThreshold ← mid

        If achievedCompression < targetCompression:
            low ← mid
        Else:
            high ← mid

        count ← count + 1

        high ← highest + 500
        low ← 0
        highest ← high
        iteration ← iteration + 1

→ bestThreshold

```

### 3.2.6.2. Algoritma

Algoritma ini digunakan untuk mencari nilai threshold terbaik yang menghasilkan persentase kompresi mendekati nilai targetCompression yang diinginkan. Proses pencarian threshold dilakukan dalam dua tahap:

#### 1. Memperluas High

Dimulai dengan nilai high = 10, program akan terus menaikkan high sebanyak 500 unit setiap kali hingga hasil kompresi (achievedCompression) melebihi atau sama dengan targetCompression. Pada tahap ini, program mencari batas atas threshold yang masih memungkinkan untuk mencapai target kompresi.

#### 2. Binary Search

Setelah nilai high ditemukan, program melakukan pencarian dengan metode biner (binary search) di antara low dan high. Nilai tengah (mid) dihitung, dan threshold tersebut diuji dengan membuat pohon QuadTree dan menghitung hasil kompresinya. Jika hasil kompresi mendekati target (selisih kurang dari 0.001), atau jika selisihnya lebih kecil dari yang sebelumnya (bestCompressionDiff), maka threshold ini disimpan sebagai kandidat terbaik (bestThreshold). Proses ini diulang maksimal 5 kali dalam dua lapis loop (iteration dan count) untuk menjaga efisiensi.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar input yang akan dikompresi.
originalSize	long	Ukuran asli file gambar (dalam byte).
low, high	double	Rentang nilai threshold untuk pencarian.
bestThreshold	double	Threshold terbaik yang ditemukan.
mid	double	Titik tengah untuk binary search.
bestCompressionDiff	double	Selisih terkecil antara kompresi tercapai dan target.
achievedCompression	double	Persentase kompresi yang diperoleh dari threshold saat ini.
diff	double	Selisih antara target kompresi dan

		kompresi tercapai.
compressed	integer	Ukuran hasil kompresi (dalam byte).
root	QuadTreeNode	Akar pohon QuadTree hasil kompresi.
iteration	integer	Jumlah iterasi outer loop pencarian.
count	integer	Jumlah iterasi inner loop binary search.
highest	double	Nilai maksimum threshold sejauh ini.
targetCompression	double	(Global Variable) Target persentase kompresi yang diinginkan

### 3.2.7. Function computeError

#### 3.2.7.1. Pseudocode

```
Function computeError(img, x, y, width, height) → real
{ Menghitung nilai error pada blok gambar (x, y, width, height)
berdasarkan metode error yang dipilih }
```

**Deklarasi:**

```
img      : Gambar (BufferedImage)
x, y    : integer // Koordinat kiri-atas blok
width, height : integer // Ukuran blok
method   : integer // Global variable, metode error
result   : real   // Nilai error yang dihitung
```

**Algoritma:**

```
switch (method)
case 1:
  result ← computeVariance(img, x, y, width, height)
case 2:
  result ← computeMAD(img, x, y, width, height)
case 3:
  result ← computeMaxDiff(img, x, y, width, height)
case 4:
  result ← computeEntropy(img, x, y, width, height)
case 5:
  result ← computeSSIM(img, x, y, width, height)
default:
  result ← computeVariance(img, x, y, width, height)
```

→ result

### 3.2.7.2. Algoritma

Fungsi computeError digunakan untuk menghitung nilai error atau kesalahan dalam satu blok gambar berdasarkan metode evaluasi yang telah dipilih sebelumnya. Blok gambar didefinisikan oleh koordinat kiri atas (x, y) serta dimensi width × height. Nilai error ini penting dalam proses kompresi QuadTree, karena menentukan apakah suatu blok bisa direpresentasikan dengan satu warna (daun) atau harus dibagi lagi (cabang).

Algoritma ini bekerja dengan melakukan *switch-case* terhadap variabel global method, yang berisi pilihan metode evaluasi yang dipilih oleh pengguna. Berdasarkan nilai ini, maka fungsi akan memanggil salah satu dari metode perhitungan error berikut:

1. **computeVariance**: Menghitung variansi warna dalam blok.
2. **computeMAD** (Mean Absolute Deviation): Mengukur rata-rata penyimpangan absolut piksel terhadap nilai rata-rata.
3. **computeMaxDiff**: Mengukur perbedaan maksimum antara piksel dalam blok.
4. **computeEntropy**: Mengukur keragaman distribusi warna dalam blok, digunakan untuk mengetahui seberapa acak blok tersebut.
5. **computeSSIM** (Structural Similarity Index): Mengukur kemiripan struktural antar blok, cocok untuk kompresi visual.

Jika nilai method tidak sesuai salah satu dari lima pilihan, maka fungsi secara default akan menggunakan computeVariance.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar input yang sedang diproses.
x, y	integer	Koordinat kiri atas blok gambar.
width	integer	Lebar blok yang akan dihitung error-nya.
height	integer	Tinggi blok yang akan dihitung error-nya.
method	integer (global)	Metode yang dipilih untuk menghitung error (1–5).

result	double	Nilai hasil perhitungan error dari blok gambar.
--------	--------	---

### 3.2.8. Function computeVariance

#### 3.2.7.1. Pseudocode

```

function computeVariance(input img : BufferedImage, input x, y, width, height : integer) → real
{ Menghitung varians rata-rata RGB dari blok (x, y, width, height) pada gambar img }

Deklarasi
    rSum, gSum, bSum : long
    count : integer
    rMean, gMean, bMean : real
    rVar, gVar, bVar : real
    color : integer
    r, g, b : real
    i, j : integer

Inisialisasi
    rSum ← 0
    gSum ← 0
    bSum ← 0
    count ← 0

// Hitung jumlah semua nilai warna
for i ← x to x + width - 1 do
    for j ← y to y + height - 1 do
        color ← img.getRGB(i, j)
        rSum ← rSum + ((color >> 16) AND 0xFF)
        gSum ← gSum + ((color >> 8) AND 0xFF)
        bSum ← bSum + (color AND 0xFF)
        count ← count + 1
    endfor
endfor

if count = 0 then
    → 0

// Hitung nilai rata-rata tiap channel
rMean ← rSum ÷ count
gMean ← gSum ÷ count
bMean ← bSum ÷ count

rVar ← 0
gVar ← 0
bVar ← 0

```

```

// Hitung total kuadrat selisih tiap warna dari rata-rata
for i ← x to x + width - 1 do
    for j ← y to y + height - 1 do
        color ← img.getRGB(i, j)
        r ← ((color >> 16) AND 0xFF) - rMean
        g ← ((color >> 8) AND 0xFF) - gMean
        b ← (color AND 0xFF) - bMean
        rVar ← rVar + (r × r)
        gVar ← gVar + (g × g)
        bVar ← bVar + (b × b)
    endfor
endfor

// Ambil rata-rata varians per kanal
rVar ← rVar ÷ count
gVar ← gVar ÷ count
bVar ← bVar ÷ count

// Kembalikan varians gabungan RGB
→ (rVar + gVar + bVar) ÷ 3

```

### 3.2.8.2. Algoritma

Fungsi computeVariance merupakan fungsi untuk mengukur error dengan menggunakan metode variansi. Pertama, fungsi menjumlahkan semua nilai kanal warna R, G, dan B dari setiap piksel dalam blok gambar. Setelah itu, nilai rata-rata dari masing-masing kanal warna dihitung dengan membagi total warna dengan jumlah piksel. Selanjutnya, fungsi menghitung selisih antara nilai warna tiap piksel dengan rata-ratanya, lalu dikuadratkan untuk mendapatkan varians. Varians dari ketiga kanal warna dijumlahkan dan dirata-ratakan, sehingga didapatkan satu nilai yang mewakili penyebaran warna dalam blok tersebut.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh pengguna
x, y	integer	Koordinat blok pada sumbu x dan y
width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.9. Function computeMAD

#### 3.2.9.1. Pseudocode

```
function computeMAD(input img : BufferedImage, input x, y, width, height : integer)
→ real
{ Menghitung nilai MAD gabungan RGB pada blok (x, y, width, height) }

Deklarasi
    rSum, gSum, bSum : long
    count : integer
    rMean, gMean, bMean : real
    madr, madg, madb : real
    color : integer
    r, g, b : real
    i, j : integer

Algoritma
rSum ← 0
gSum ← 0
bSum ← 0
count ← 0

// Hitung jumlah warna untuk rata-rata
for i ← x to min(x + width - 1, img.width - 1) do
    for j ← y to min(y + height - 1, img.height - 1) do
        color ← img.getRGB(i, j)
        rSum ← rSum + ((color >> 16) AND 0xFF)
        gSum ← gSum + ((color >> 8) AND 0xFF)
        bSum ← bSum + (color AND 0xFF)
        count ← count + 1
    endfor
endfor

if count = 0 then
    → 0

// Hitung rata-rata warna
rMean ← rSum ÷ count
gMean ← gSum ÷ count
bMean ← bSum ÷ count

madr ← 0
madg ← 0
madb ← 0

// Hitung deviasi absolut
for i ← x to min(x + width - 1, img.width - 1) do
    for j ← y to min(y + height - 1, img.height - 1) do
        color ← img.getRGB(i, j)
```

```

r ← ((color >> 16) AND 0xFF) - rMean
g ← ((color >> 8) AND 0xFF) - gMean
b ← (color AND 0xFF) - bMean
madr ← madr + abs(r)
madg ← madg + abs(g)
madb ← madb + abs(b)
endfor
endfor

// Hitung rata-rata MAD tiap kanal
madr ← madr ÷ count
madg ← madg ÷ count
madb ← madb ÷ count

// Kembalikan nilai MAD gabungan
→ (madr + madg + madb) ÷ 3

```

### 3.2.9.2. Algoritma

Fungsi computeMAD menghitung tingkat penyimpangan rata-rata absolut warna dari blok gambar pada area (x, y, width, height). Pertama, fungsi ini melakukan perhitungan rata-rata nilai R, G, dan B dari semua piksel dalam blok. Setelah mendapatkan rata-ratanya, fungsi menghitung selisih absolut antara setiap nilai piksel dengan rata-ratanya, lalu dijumlahkan dan dirata-ratakan untuk masing-masing kanal warna. Nilai akhir yang dikembalikan adalah rata-rata dari ketiga MAD kanal, yang menunjukkan seberapa besar rata-rata penyimpangan warna dalam blok tersebut.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh pengguna
x, y	integer	Koordinat blok pada sumbu x dan y
width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.10. Function computeMaxDiff

#### 3.2.10.1. Pseudocode

```

function computeMaxDiff(input img : BufferedImage, input x, y, width, height : integer) → real
{ Mengembalikan nilai rata-rata selisih max-min dari tiap kanal RGB }

```

**Deklarasi**

```
rMin, gMin, bMin ← 255  
rMax, gMax, bMax ← 0  
color, r, g, b ← integer  
i, j ← integer
```

**Algoritma**

```
// Cari nilai maksimum dan minimum tiap kanal warna  
for i ← x to min(x + width - 1, img.width - 1) do  
    for j ← y to min(y + height - 1, img.height - 1) do  
        color ← img.getRGB(i, j)  
        r ← (color >> 16) AND 0xFF  
        g ← (color >> 8) AND 0xFF  
        b ← color AND 0xFF  
  
        rMin ← min(rMin, r)  
        gMin ← min(gMin, g)  
        bMin ← min(bMin, b)  
  
        rMax ← max(rMax, r)  
        gMax ← max(gMax, g)  
        bMax ← max(bMax, b)  
endfor  
endfor  
  
// Hitung selisih max-min tiap kanal  
dR ← rMax - rMin  
dG ← gMax - gMin  
dB ← bMax - bMin  
  
// Rata-rata dari ketiga selisih  
→ (dR + dG + dB) ÷ 3.0
```

**3.2.10.2. Algoritma**

Fungsi computeMaxDiff digunakan untuk menghitung perbedaan maksimum warna dalam sebuah blok gambar. Algoritma ini bekerja dengan mencari nilai minimum dan maksimum untuk setiap kanal warna dari seluruh piksel dalam blok. Selanjutnya, dihitung selisih antara nilai maksimum dan minimum untuk masing-masing kanal. Nilai akhir yang dikembalikan adalah rata-rata dari ketiga selisih tersebut, yang mencerminkan seberapa besar variasi warna dalam blok.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh

		pengguna
x, y	integer	Koordinat blok pada sumbu x dan y
width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.11. Function averageColor

#### 3.2.11.1. Pseudocode

```

function averageColor(input img : BufferedImage, input x, y, width, height : integer)
→ integer
{ Mengembalikan warna rata-rata dalam blok sebagai nilai RGB integer }

Deklarasi
    r, g, b ← 0 (long)
    count ← 0
    i, j ← integer
    color ← integer
    red, green, blue ← integer

Algoritma
// Menjumlahkan nilai tiap kanal warna
for i ← x to min(x + width - 1, img.width - 1) do
    for j ← y to min(y + height - 1, img.height - 1) do
        color ← img.getRGB(i, j)
        red ← (color >> 16) AND 0xFF
        green ← (color >> 8) AND 0xFF
        blue ← color AND 0xFF

        r ← r + red
        g ← g + green
        b ← b + blue
        count ← count + 1
    endfor
endfor

// Hindari pembagian nol
if count = 0 then
    → 0
endif

// Hitung rata-rata dan gabungkan ke format RGB
r ← r ÷ count
g ← g ÷ count
b ← b ÷ count

```

```
→ (r << 16) OR (g << 8) OR b
```

### 3.2.11.2. Algoritma

Fungsi averageColor digunakan untuk menghitung rata-rata warna dari sebuah blok gambar pada posisi dan ukuran tertentu. Fungsi ini menjumlahkan seluruh nilai RGB dari tiap piksel dalam blok, lalu membaginya dengan jumlah total piksel untuk mendapatkan nilai rata-rata dari masing-masing kanal warna. Setelah diperoleh nilai rata-rata tiap kanal, hasilnya digabung menjadi satu integer warna (RGB 24-bit) dan dikembalikan. Jika blok tidak memiliki piksel (misal karena ukuran nol), maka fungsi akan mengembalikan 0.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh pengguna
x, y	integer	Koordinat blok pada sumbu x dan y
width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.12. Function computeEntropy

#### 3.2.12.1. Pseudocode

```
function computeEntropy(input img : BufferedImage, input x, y, width, height : integer) → double
{ Mengembalikan nilai entropi rata-rata dari kanal R, G, B dalam blok }
```

#### Deklarasi:

```
rVal[0..255], gVal[0..255], bVal[0..255] ← 0 (array of integer)
rProb[0..255], gProb[0..255], bProb[0..255] ← 0.0 (array of double)
count ← 0
i, j, color ← integer
r, g, b ← integer
rEntro, gEntro, bEntro ← 0.0 (double)
```

#### Algoritma

```
// Hitung histogram (frekuensi nilai warna)
for i ← x to min(x + width - 1, img.width - 1) do
    for j ← y to min(y + height - 1, img.height - 1) do
        color ← img.getRGB(i, j)
        r ← (color >> 16) AND 0xFF
```

```

g ← (color >> 8) AND 0xFF
b ← color AND 0xFF
rVal[r] ← rVal[r] + 1
gVal[g] ← gVal[g] + 1
bVal[b] ← bVal[b] + 1
count ← count + 1
endfor
endfor

if count = 0 then
    → 0.0
endif

// Hitung probabilitas dari histogram
for i ← 0 to 255 do
    rProb[i] ← rVal[i] ÷ count
    gProb[i] ← gVal[i] ÷ count
    bProb[i] ← bVal[i] ÷ count
endfor

// Hitung entropi dari probabilitas
for i ← 0 to 255 do
    if rProb[i] > 0 then rEntro ← rEntro - (rProb[i] × log2(rProb[i])) endif
    if gProb[i] > 0 then gEntro ← gEntro - (gProb[i] × log2(gProb[i])) endif
    if bProb[i] > 0 then bEntro ← bEntro - (bProb[i] × log2(bProb[i])) endif
endfor

→ (rEntro + gEntro + bEntro) ÷ 3

```

### 3.2.12.2. Algoritma

Fungsi computeEntropy digunakan untuk menghitung entropi warna dari sebuah blok gambar. Entropi mengukur seberapa acak atau kompleks distribusi nilai warna dalam blok. Fungsi ini memulai dengan menghitung frekuensi (histogram) dari nilai-nilai kanal merah, hijau, dan biru. Lalu, fungsi menghitung probabilitas kemunculan tiap nilai dari hasil frekuensi tersebut. Akhirnya, menggunakan rumus Shannon entropy, fungsi menjumlahkan  $-p \cdot \log_2(p)$  untuk tiap nilai kanal dan mengambil rata-rata entropi dari ketiga kanal. Jika blok tidak memiliki piksel, entropi dianggap nol.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh pengguna
x, y	integer	Koordinat blok pada sumbu x dan y

width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.13. Function computeSSIM

#### 3.2.13.1. Pseudocode

```

function computeSSIM(input img: BufferedImage, input x: int, y, width, height: integer) → double
{ Mengembalikan nilai SSIM dari blok gambar terhadap blok rata-rata }

Deklarasi
    ryMean, gyMean, byMean ← nilai rata-rata R, G, B dari averageColor(img, x,
y, width, height)
    C1 ← 6.5025
    C2 ← 58.5225
    rxSum, gxSum, bxSum ← 0
    count ← 0

Algoritma
// Hitung rata-rata piksel aktual dalam blok
for i ← x to min(x + width - 1, img.width - 1) do
    for j ← y to min(y + height - 1, img.height - 1) do
        color ← img.getRGB(i, j)
        rxSum += (color >> 16) AND 0xFF
        gxSum += (color >> 8) AND 0xFF
        bxSum += color AND 0xFF
        count ← count + 1
    endfor
endfor

if count = 0 then
    → 0
endif

rxMean ← rxSum ÷ count
gxMean ← gxSum ÷ count
bxMean ← bxSum ÷ count

// Hitung variansi
rxVar, gxVar, bxVar ← 0
for i ← x to x + width - 1 do
    for j ← y to y + height - 1 do
        color ← img.getRGB(i, j)
        rx ← (color >> 16) AND 0xFF
        gx ← (color >> 8) AND 0xFF
        bx ← color AND 0xFF
        rxVar += (rx - rxMean)^2

```

```

        gxVar += (gx - gxMean)^2
        bxVar += (bx - bxMean)^2
    endfor
endfor
rxvar /= count;
gxvar /= count;
bxvar /= count;

// Hitung SSIM untuk tiap kanal warna
rSSIM ← ((2 × rxMean × ryMean + C1) × C2) ÷ ((rxMean2 + ryMean2 + C1) × (rxVar + C2))
gSSIM ← ((2 × gxMean × gyMean + C1) × C2) ÷ ((gxMean2 + gyMean2 + C1) × (gxVar + C2))
bSSIM ← ((2 × bxMean × byMean + C1) × C2) ÷ ((bxMean2 + byMean2 + C1) × (bxVar + C2))

// Gabungkan SSIM RGB dengan bobot luminansi
→ 1-(0.2989 × rSSIM + 0.5870 × gSSIM + 0.1140 × bSSIM)

```

### 3.2.13.2. Algoritma

Fungsi `computeSSIM` digunakan untuk menghitung Structural Similarity Index (SSIM) antara blok gambar dan versi blok rata-ratanya (yang dianggap sebagai referensi atau "ideal"). Dalam kasus ini, blok ideal adalah blok dari gambar yang belum dikompresi, sedangkan blok rata-rata adalah blok dari gambar yang sudah dikompresi. SSIM mengukur seberapa mirip struktur dari sebuah blok gambar terhadap blok ideal dengan mempertimbangkan luminansi, kontras, dan struktur. Langkah pertama adalah menghitung nilai rata-rata RGB dari blok sebagai referensi. Kemudian, fungsi menghitung rata-rata aktual dari blok, serta variansnya. SSIM dihitung untuk setiap kanal warna menggunakan rumus SSIM standar dengan konstanta C1 dan C2 untuk stabilisasi. Hasil akhirnya merupakan kombinasi dari SSIM merah, hijau, dan biru berdasarkan bobot luminansi.

Parameter	Type	Keterangan
img	Buffered Image	Objek gambar yang di input oleh pengguna
x, y	integer	Koordinat blok pada sumbu x dan y
width	integer	Lebar dari gambar yang sedang ditinjau
height	integer	Tinggi dari gambar yang sedang ditinjau

### 3.2.14. Procedure getSerializedSize

#### 3.2.14.1. Pseudocode

```
Function getSerializedSize(root: QuadTreeNode) → integer  
{ Menghitung ukuran hasil serialisasi dari pohon QuadTree }
```

**Deklarasi:**

```
size : ByteArrayOutputStream  
oos : ObjectOutputStream  
length : integer
```

**Algoritma:**

1. Buat objek stream `size` untuk menampung data dalam bentuk byte
2. Buat objek stream `oos` untuk menulis objek ke dalam `size`
3. Tulis objek `root` (QuadTreeNode) ke dalam `oos`
4. Flush data agar semua ditulis ke stream
5. Tutup stream `oos`
6. Ambil byte array dari stream `size`
7. Hitung panjang array tersebut sebagai `length`
8. → length

#### 3.2.14.2. Algoritma

Algoritma getSerializedSize digunakan untuk menghitung ukuran hasil kompresi dari struktur data pohon QuadTreeNode dengan cara melakukan serialisasi objek root yang merupakan akar dari QuadTreeNode. Proses dimulai dengan membuat objek ByteArrayOutputStream, yang berfungsi sebagai penampung aliran data dalam bentuk byte. Kemudian, ObjectOutputStream digunakan untuk menulis objek Java ke dalam ByteArrayOutputStream. Objek root diserialisasi ke dalam bentuk byte menggunakan metode writeObject, dan setelah semua data ditulis, stream dibersihkan (flush) dan ditutup (close) agar tidak ada data yang tertinggal dalam buffer. Terakhir, seluruh data dalam stream diubah menjadi array byte, dan panjang array tersebut dihitung untuk mendapatkan ukuran akhir dari data hasil serialisasi. Nilai ini menunjukkan seberapa besar ukuran memori dari struktur QuadTree yang telah dikompresi.

Parameter	Type	Keterangan
root	QuadTreeNode	Objek yang merupakan akar dari pohon QuadTreeNode yang mewakili kompresi gambar

size	ByteArrayOutputStream	Buffer untuk menampung hasil serialisasi dalam bentuk byte.
oos	ObjectOutputStream	Stream untuk menulis objek Java ke dalam bentuk byte.
size.toByteArray()	byte[]	Array byte yang mewakili objek setelah diserialisasi.
size.toByteArray().length	integer	Nilai panjang array byte, yaitu ukuran akhir dari hasil kompresi.

### 3.2.15. Procedure printTableHeader, printRow, and printTableFooter

#### 3.2.15.1. Pseudocode

```

Procedure printTableHeader()
    Output "+-----+-----+"
    Output "| Parameter | Nilai |"
    Output "+-----+-----+"
EndProcedure

Procedure printRow(parameter : String, value : String, color : String)
    Output " " + parameter (dengan format rata kiri, panjang 44, dan berwarna menggunakan 'color') +
           " " + value (rata kiri, panjang 24) + " "
EndProcedure

Procedure printTableFooter()
    Output "+-----+-----+"
EndProcedure

```

#### 3.2.15.2. Algoritma

Algoritma printTableHeader, printRow, dan printTableFooter digunakan untuk mencetak tabel informasi ke terminal dengan tampilan yang rapi dan estetis menggunakan 3 bagian utama, yaitu header, baris isi, dan footer. Pewarnaan teks dilakukan dengan kode warna ANSI yang memungkinkan teks terminal ditampilkan dalam warna tertentu untuk menekankan informasi.

##### 1. Header Tabel (printTableHeader)

Bagian ini mencetak batas atas tabel dan judul kolom. Judul kolom terdiri dari "Parameter" dan "Nilai", yang masing-masing dibatasi oleh garis horizontal. Ini membantu pengguna langsung memahami isi tabel.

##### 2. Isi Tabel (printRow)

Setiap baris diisi dengan dua nilai: nama parameter dan nilainya. Format lebar kolom ditentukan agar rata dan sejajar (parameter sepanjang 44 karakter, nilai sepanjang 24 karakter). Pewarnaan teks untuk nama parameter menggunakan ANSI escape code yang diberikan melalui parameter color. Ini memungkinkan pemisahan visual antar baris dan memberi penekanan pada informasi penting.

### 3. Footer Tabel (printTableFooter)

Bagian akhir tabel mencetak batas bawah untuk menunjukkan bahwa tidak ada data tambahan, memberikan kesan penutup yang rapi.

## BAB 4

# IMPLEMENTASI DAN PENGUJIAN

### 4.1. Implementasi

#### 4.1.1. QuadTreeCompression.java

##### 4.1.1.1. Prosedur Utama (Main)

```
public static void main(String[] args) throws Exception {
    Scanner scanner = new Scanner(System.in);
    BufferedImage img = null;
    String imagePath;

    printTableHeader();

    // Loop untuk validasi gambar
    while (true) {
        System.out.print("Masukkan nama gambar (berserta ekstensi .jpg/.jpeg/.png): ");
        String imageName = scanner.nextLine();
        imagePath = imageName;

        File imageFile = new File(imagePath);

        // Cek apakah file ada
        if (!imageFile.exists() || imageFile.isDirectory()) {
            System.out.println("File tidak ditemukan. Silakan masukkan nama gambar yang benar.");
            continue;
        }

        try {
            img = ImageIO.read(imageFile);

            // Jika gambar tidak valid
            if (img == null) {
                System.out.println("Format gambar tidak didukung atau file rusak. Silakan masukkan gambar lain.");
                continue;
            }
            printRow(parameter:"Nama Gambar", imageName, ANSI_GREEN);
            // Jika gambar valid, keluar dari loop
            break;
        } catch (IOException e) {
            System.out.println("Terjadi kesalahan saat membaca gambar: " + e.getMessage());
        }
    }

    File inputFile = new File (imagePath);
    long originalSize = inputFile.length();

    System.out.print("Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): ");
    method = scanner.nextInt();
    printRow(parameter:"Metode Error", String.valueOf(method), ANSI_YELLOW);

    System.out.print("Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): ");
    targetCompression = scanner.nextDouble();
    printRow(parameter:"Target Kompresi", String.valueOf(targetCompression), ANSI_BLUE);

    if (targetCompression == 0){
        isMinBlock = true;
        System.out.print("Masukkan threshold: ");
        threshold = scanner.nextDouble();
        printRow(parameter:"Threshold", String.valueOf(threshold), ANSI_BLUE);
    }
}
```

```

        System.out.print("Masukkan ukuran blok minimum: ");
        minBlockSize = scanner.nextInt();
        printRow(parameter:"Blok Minimum", String.valueOf(minBlockSize), ANSI_BLUE);
    } else{
        isMinBlock = false;
        long thresholdStart = System.nanoTime();
        threshold = findBestThreshold(img, originalSize);
        thresholdTime = System.nanoTime() - thresholdStart;
        printRow(parameter:"Threshold Otomatis", String.format(format:".2f", threshold), ANSI_BLUE);
    }
    System.out.print("Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): ");
    String outputName = scanner.next();
    printRow(parameter:"Output Gambar", outputName, ANSI_GREEN);

    System.out.print("Masukkan nama file GIF hasil (akhiri dengan .gif): ");
    String gifName = scanner.next();
    printRow(parameter:"Output GIF", gifName, ANSI_GREEN);

    printTableFooter();

    String outputPath = outputName;
    String gifPath = gifName;

    startTime = System.nanoTime();
    currentFrame = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_ARGB);
    Graphics2D g = currentFrame.createGraphics();
    g.setColor(new Color(averageColor(img, x:0, y:0, img.getWidth(), img.getHeight())));
    g.fillRect(x:0, y:0, img.getWidth(), img.getHeight());
    g.dispose();

    QuadTreeNode root = buildQuadTreeBFS(img);
    saveGif(gifPath, frames, delayMs:500); // delay per frame: 200ms
    Long executionTime = System.nanoTime() - startTime;

    BufferedImage compressedImage = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_RGB);
    drawQuadTree(compressedImage, root);
    ImageIO.write(compressedImage, formatName:"png", new File(outputPath));

    int compressedSize = getSerializedSize(root);
    double compressionPercentage = 100.0 - ((double) compressedSize / originalSize * 100);

    System.out.println("\n" + ANSI_GREEN + "Hasil Kompresi:" + ANSI_RESET);
    printTableHeader();
    printRow(parameter:"Ukuran Awal", originalSize + " bytes", ANSI_YELLOW);
    printRow(parameter:"Ukuran Akhir", compressedSize + " bytes", ANSI_YELLOW);
    printRow(parameter:"Kompresi", String.format(format:".2f%%", compressionPercentage), ANSI_GREEN);
    printRow(parameter:"Maklumat Eksekusi", String.format(format:".2f ms", executionTime / 1e6), ANSI_CYAN);
    printRow(parameter:"Kedalaman Maksimal", String.valueOf(maxDepth), ANSI_CYAN);
    printRow(parameter:"Total Simpul", String.valueOf(totalNodes), ANSI_CYAN);
    printTableFooter();

    System.out.println("Gambar hasil disimpan di: " + outputPath);
    System.out.println("GIF hasil disimpan di: " + gifPath);
}

```

#### 4.1.1..2. Fungsi copyOf()

```

private static BufferedImage copyOf(BufferedImage img) {
    BufferedImage copy = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_ARGB);
    Graphics2D g = copy.createGraphics();
    g.drawImage(img, x:0, y:0, observer:null);
    g.dispose();
    return copy;
}

```

#### 4.1.3..3. Fungsi buildQuadTreeBFS()

```

private static QuadTreeNode buildQuadTreeBFS(BufferedImage img) {
    Queue<QuadTreeNode> queue = new LinkedList<>();
    QuadTreeNode root = new QuadTreeNode(x:0, y:0, img.getWidth(), img.getHeight(),
                                         averageColor(img, x:0, y:0, img.getWidth(), img.getHeight()), isLeaf:false, depth:0);
    queue.offer(root);

    currentFrame = new BufferedImage(img.getWidth(), img.getHeight(), BufferedImage.TYPE_INT_ARGB);
    frames.clear();

    // Frame awal (satu warna)
    Graphics2D gInit = currentFrame.createGraphics();
    gInit.setColor(new Color(root.color));
    gInit.fillRect(x:0, y:0, img.getWidth(), img.getHeight());
    gInit.dispose();
    frames.add(copyOf(currentFrame));

    List<QuadTreeNode> currentLeaves = new ArrayList<>();

    while (!queue.isEmpty()) {
        int size = queue.size();
        List<QuadTreeNode> newChildren = new ArrayList<>();

        for (int i = 0; i < size; i++) {
            QuadTreeNode node = queue.poll();
            double err = computeError(img, node.x, node.y, node.width, node.height);
            int halfWidth = node.width / 2;
            int halfHeight = node.height / 2;
            // System.out.println("ini SSIM: " +err);
            if((isMinBlock && (halfWidth*halfHeight < minBlockSize || err < threshold)) || (!isMinBlock && (err < threshold))){
                node.isLeaf = true;
                totalNodes++;
                currentLeaves.add(node);
            }else {
                QuadTreeNode[] children = new QuadTreeNode[4];
                children[0] = new QuadTreeNode(node.x, node.y, halfWidth, halfHeight,
                                              averageColor(img, node.x, node.y, halfWidth, halfHeight), isLeaf:false, node.depth+1);
                children[1] = new QuadTreeNode(node.x + halfWidth, node.y, node.width - halfWidth, halfHeight,
                                              averageColor(img, node.x + halfWidth, node.y, node.width - halfWidth, halfHeight), isLeaf:false, node.depth+1);
                children[2] = new QuadTreeNode(node.x, node.y + halfHeight, halfWidth, node.height - halfHeight,
                                              averageColor(img, node.x, node.y + halfHeight, halfWidth, node.height - halfHeight), isLeaf:false, node.depth+1);
                children[3] = new QuadTreeNode(node.x + halfWidth, node.y + halfHeight, node.width - halfWidth,
                                              averageColor(img, node.x + halfWidth, node.y + halfHeight, node.width - halfWidth,
                                                          node.height - halfHeight),
                                              isLeaf:false, node.depth+1);
            }
            node.children = children;
            Collections.addAll(newChildren, children);
        }
        if (!newChildren.isEmpty()) {
            Graphics2D g = currentFrame.createGraphics();
            // gambar semua leaf
            for (QuadTreeNode leaf : currentLeaves) {
                g.setColor(new Color(leaf.color));
                g.fillRect(leaf.x, leaf.y, leaf.width, leaf.height);
            }
            // gambar semua children baru
            for (QuadTreeNode child : newChildren) {
                maxDepth = Math.max(maxDepth, child.depth);
                g.setColor(new Color(child.color));
                g.fillRect(child.x, child.y, child.width, child.height);
            }
            g.dispose();
            frames.add(copyOf(currentFrame));
            queue.addAll(newChildren);
        }
    }
    return root;
}

```

#### 4.1.1.4. Fungsi buildQuadTreeBFSwithThreshold()

```

private static QuadTreeNode buildQuadTreeBFSwithThreshold(BufferedImage img, double thresholdVal) {
    Queue<QuadTreeNode> queue = new LinkedList<>();
    QuadTreeNode root = new QuadTreeNode(x:0, y:0, img.getWidth(), img.getHeight(),
                                         averageColor(img, x:0, y:0, img.getWidth(), img.getHeight()), isLeaf:false, depth:0);
    queue.offer(root);

    List<QuadTreeNode> currentLeaves = new ArrayList<>();

    while (!queue.isEmpty()) {
        int size = queue.size();
        List<QuadTreeNode> newChildren = new ArrayList<>();

        for (int i = 0; i < size; i++) {
            QuadTreeNode node = queue.poll();
            double err = computeError(img, node.x, node.y, node.width, node.height);

            if (err < thresholdVal) {
                node.isLeaf = true;
                currentLeaves.add(node);
            } else {
                int halfWidth = node.width / 2;
                int halfHeight = node.height / 2;
                QuadTreeNode[] children = new QuadTreeNode[4];
                children[0] = new QuadTreeNode(node.x, node.y, halfWidth, halfHeight,
                                              averageColor(img, node.x, node.y, halfWidth, halfHeight), isLeaf:false, node.depth+1);
                children[1] = new QuadTreeNode(node.x + halfWidth, node.y, node.width - halfWidth, halfHeight,
                                              averageColor(img, node.x + halfWidth, node.y, node.width - halfWidth, halfHeight), isLeaf:false, node.depth+1);
                children[2] = new QuadTreeNode(node.x, node.y + halfHeight, halfWidth, node.height - halfHeight,
                                              averageColor(img, node.x, node.y + halfHeight, halfWidth, node.height - halfHeight), isLeaf:false, node.depth+1);
                children[3] = new QuadTreeNode(node.x + halfWidth, node.y + halfHeight, node.width - halfWidth,
                                              node.height - halfHeight,
                                              averageColor(img, node.x + halfWidth, node.y + halfHeight, node.width - halfWidth,
                                                          node.height - halfHeight),
                                              isLeaf:false, node.depth+1);

                node.children = children;
                Collections.addAll(newChildren, children);
            }
        }

        if (!newChildren.isEmpty())
            queue.addAll(newChildren);
    }

    return root;
}

```

#### 4.1.1.5. Prosedur saveGif()

```

private static void saveGif(String path, List<BufferedImage> frames, int delayMs) throws IOException {
    ImageOutputStream output = new FileImageOutputStream(new File(path));
    GifSequenceWriter writer = new GifSequenceWriter(output, BufferedImage.TYPE_INT_ARGB, delayMs, loop:true);
    int i = 1;
    for (BufferedImage frame : frames) {
        writer.writeToSequence(frame);
        //ImageIO.write(frame, "png", new File("frame_" + i + ".png"));
        i++;
    }
    writer.close();
    output.close();
}

```

#### 4.1.1.6. Prosedur drawQuadTree()

```

private static void drawQuadTree(BufferedImage img, QuadTreeNode node) {
    if (node.isLeaf) {
        Graphics2D g = img.createGraphics();
        g.setColor(new Color(node.color));
        g.fillRect(node.x, node.y, node.width, node.height);
        g.dispose();
    } else {
        for (QuadTreeNode child : node.children) {
            if (child != null) drawQuadTree(img, child);
        }
    }
}

```

#### 4.1.1.7. Fungsi findBestThreshold()

```

private static double findBestThreshold(BufferedImage img, Long originalSize) throws IOException {
    double low = 0;
    double high = 10;
    double bestCompressionDiff = Double.MAX_VALUE;
    double achievedCompression = 0;
    int iteration = 0;

    // System.out.println("Target Compression: " + targetCompression);

    // Step 1: Perluas high jika kompresi masih negatif
    while (true) {
        QuadTreeNode root = buildQuadTreeBFSwithThreshold(img, high);
        int compressed = getSerializedSize(root);
        achievedCompression = 1.0 - ((double) compressed / originalSize);
        // System.out.printf("[EXTEND-HIGH] Threshold=%2f | Compressed=%d | Achieved=%4f\n", high, compressed, achievedCompression);

        if (achievedCompression >= targetCompression) break;
        high += 500;
    }
    double bestThreshold = high;
    double highest = high;
    double diff = Math.abs(achievedCompression - targetCompression);

    // Step 2: Binary search
    while (iteration <= 5 && diff > 0.001) {
        // System.out.println("achieved found: " + achievedCompression);
        // System.out.println("target: " + targetCompression);
        // System.out.println("diff: " + diff);
        int count = 0;
        while (diff > 0.001 && count <= 5) {
            double mid = (low + high) / 2;
            QuadTreeNode root = buildQuadTreeBFSwithThreshold(img, mid);
            int compressed = getSerializedSize(root);
            achievedCompression = 1.0 - ((double) compressed / originalSize);

            diff = Math.abs(achievedCompression - targetCompression);
            if (achievedCompression >= 0 && diff < bestCompressionDiff) {
                bestCompressionDiff = diff;
                bestThreshold = mid;
            }
        }

        if (achievedCompression < targetCompression) {
            low = mid;
        } else {
            high = mid;
        }
        count++;
        // System.out.printf("Mid: %2f | Achieved: %.4f | Diff: %.4f | Best: %.2f\n", mid, achievedCompression, diff, bestThreshold);
    }
    // System.out.println("-----");
    high = highest + 500;
    low = 0;
    highest = high;
    iteration++;
}

// System.out.println("Best threshold found: " + bestThreshold);
return bestThreshold;
}

```

#### 4.1.1.8. Fungsi computeError()

```
private static double computeError(BufferedImage img, int x, int y, int width, int height) {
    switch (method) {
        case 1: return computeVariance(img, x, y, width, height);
        case 2: return computeMAD(img, x, y, width, height);
        case 3: return computeMaxDiff(img, x, y, width, height);
        case 4: return computeEntropy(img, x, y, width, height);
        case 5: return computeSSIM(img, x, y, width, height);
        default: return computeVariance(img, x, y, width, height);
    }
}
```

#### 4.1.1.9. Fungsi computeVariance()

```
private static double computeVariance(BufferedImage img, int x, int y, int width, int height) {
    long rSum = 0, gSum = 0, bSum = 0;
    int count = 0;

    // if (width * height > minBlockSize){
    //     return 0;
    // }
    // Hitung rata-rata tiap kanal warna
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            int color = img.getRGB(i, j);
            rSum += (color >> 16) & 0xFF;
            gSum += (color >> 8) & 0xFF;
            bSum += color & 0xFF;
            count++;
        }
    }

    if (count == 0) return 0; // Hindari pembagian dengan nol

    double rMean = (double) rSum / count;
    double gMean = (double) gSum / count;
    double bMean = (double) bSum / count;

    // Hitung varians tiap kanal warna
    double rVar = 0, gVar = 0, bVar = 0;

    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            int color = img.getRGB(i, j);
            double r = ((color >> 16) & 0xFF) - rMean;
            double g = ((color >> 8) & 0xFF) - gMean;
            double b = (color & 0xFF) - bMean;
            rVar += r * r;
            gVar += g * g;
            bVar += b * b;
        }
    }

    rVar /= count;
    gVar /= count;
    bVar /= count;

    // Hitung varians RGB gabungan
    return (rVar + gVar + bVar) / 3.0;
}
```

#### 4.1.1.10. Fungsi computeMAD()

```
private static double computeMAD(BufferedImage img, int x, int y, int width, int height) {
    long rSum = 0, gSum = 0, bSum = 0;
    int count = 0;

    // Hitung rata-rata tiap kanal warna
    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            rSum += (color >> 16) & 0xFF;
            gSum += (color >> 8) & 0xFF;
            bSum += color & 0xFF;
            count++;
        }
    }

    if (count == 0) return 0;

    double rMean = (double) rSum / count;
    double gMean = (double) gSum / count;
    double bMean = (double) bSum / count;

    // Hitung varians tiap kanal warna
    double madr = 0, madg = 0, madb = 0;

    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            double r = ((color >> 16) & 0xFF) - rMean;
            double g = ((color >> 8) & 0xFF) - gMean;
            double b = (color & 0xFF) - bMean;
            madr += Math.abs(r);
            madg += Math.abs(g);
            madb += Math.abs(b);
        }
    }

    madr /= count;
    madg /= count;
    madb /= count;

    // Hitung varians RGB gabungan
    return (madr + madg + madb) / 3.0;
}
```

#### 4.1.1.11. Fungsi computeMaxDiff()

```

private static double computeMaxDiff(BufferedImage img, int x, int y, int width, int height) {
    int rMin = 255, gMin = 255, bMin = 255;
    int rMax = 0, gMax = 0, bMax = 0;

    // Hitung nilai min dan max tiap kanal warna dalam blok
    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            int r = (color >> 16) & 0xFF;
            int g = (color >> 8) & 0xFF;
            int b = color & 0xFF;

            rMin = Math.min(rMin, r);
            gMin = Math.min(gMin, g);
            bMin = Math.min(bMin, b);

            rMax = Math.max(rMax, r);
            gMax = Math.max(gMax, g);
            bMax = Math.max(bMax, b);
        }
    }

    // Hitung selisih max-min tiap kanal
    int dR = rMax - rMin;
    int dG = gMax - gMin;
    int dB = bMax - bMin;

    // Hitung D_RGB sesuai rumus
    return (dR + dG + dB) / 3.0;
}

```

#### 4.1.1.12. Fungsi averageColor()

```

private static int averageColor(BufferedImage img, int x, int y, int width, int height) {
    long r = 0, g = 0, b = 0;
    int count = 0;

    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            r += (color >> 16) & 0xFF;
            g += (color >> 8) & 0xFF;
            b += color & 0xFF;
            count++;
        }
    }

    if (count == 0) return 0;
    r /= count;
    g /= count;
    b /= count;

    return (int) ((r << 16) | (g << 8) | b);
}

```

#### 4.1.1.13. Fungsi computeEntropy()

```

private static double computeEntropy(BufferedImage img, int x, int y, int width, int height) {
    int r = 0, g = 0, b = 0, count = 0;
    int[] rVal = new int[256];
    int[] gVal = new int[256];
    int[] bVal = new int[256];
    double[] rProb = new double[256];
    double[] gProb = new double[256];
    double[] bProb = new double[256];
    double rEntro = 0, gEntro = 0, bEntro = 0;

    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            r = (color >> 16) & 0xFF;
            g = (color >> 8) & 0xFF;
            b = color & 0xFF;
            rVal[r]++;
            gVal[g]++;
            bVal[b]++;
            count++;
        }
    }
    if (count == 0) return 0;
    for (int i = 0; i < 256; i++) {
        rProb[i] = (double) rVal[i]/count;
        gProb[i] = (double) gVal[i]/count;
        bProb[i] = (double) bVal[i]/count;
    }

    for (int i = 0; i < 256; i++) {
        if (rProb[i] > 0) rEntro -= rProb[i] * (Math.log(rProb[i]) / Math.log(a:2));
        if (gProb[i] > 0) gEntro -= gProb[i] * (Math.log(gProb[i]) / Math.log(a:2));
        if (bProb[i] > 0) bEntro -= bProb[i] * (Math.log(bProb[i]) / Math.log(a:2));
    }

    return (rEntro+gEntro+bEntro)/3;
}

```

#### 4.1.1.14. Fungsi computeSSIM()

```

private static double computeSSIM(BufferedImage img, int x, int y, int width, int height) {
    Color avgColor = new Color(averageColor(img, x, y, width, height));
    int rxMean = avgColor.getRed();
    int gyMean = avgColor.getGreen();
    int byMean = avgColor.getBlue();
    // C = (KL)^2, K = 0.01, K = 0.03
    double C1 = 6.5025, C2 = 58.5225;
    int rx = 0, gx = 0, bx = 0;
    int rxSum = 0, gxSum = 0, bxSum = 0;
    int count = 0;

    for (int i = x; i < x + width && i < img.getWidth(); i++) {
        for (int j = y; j < y + height && j < img.getHeight(); j++) {
            int color = img.getRGB(i, j);
            rx = (color >> 16) & 0xFF;
            gx = (color >> 8) & 0xFF;
            bx = color & 0xFF;
            rxSum += rx;
            gxSum += gx;
            bxSum += bx;
            count++;
        }
    }

    if (count == 0) return 0;
    double rxMean = (double) rxSum / count;
    double gxMean = (double) gxSum / count;
    double bxMean = (double) bxSum / count;

    double rxVar = 0, gxVar = 0, bxVar = 0;
    for (int i = x; i < x + width; i++) {
        for (int j = y; j < y + height; j++) {
            int color = img.getRGB(i, j);
            double r = ((color >> 16) & 0xFF) - rxMean;
            double g = ((color >> 8) & 0xFF) - gxMean;
            double b = (color & 0xFF) - bxMean;
            rxVar += r * r;
            gxVar += g * g;
            bxVar += b * b;
        }
    }

    rxVar /= count;
    gxVar /= count;
    bxVar /= count;

    double rSSIM = ((2*rxMean*ryMean+C1)*C2)/((rxMean*rxMean+ryMean*ryMean+C1)*(rxVar+C2));
    double gSSIM = ((2*gyMean*gyMean+C1)*C2)/((gxMean*gxMean+gyMean*gyMean+C1)*(gxVar+C2));
    double bSSIM = ((2*bxMean*byMean+C1)*C2)/((bxMean*bxMean+byMean*byMean+C1)*(bxVar+C2));
    //System.out.print("Ini hasil: ");
    //System.out.println(0.299*rSSIM+0.587*gSSIM+0.114*bSSIM);
    return 1-(0.299*rSSIM+0.587*gSSIM+0.114*bSSIM);
}

```

#### 4.1.1.1.15. Fungsi getSerializedSize()

```
public static int getSerializedSize(QuadTreeNode root) throws IOException{
    ByteArrayOutputStream size = new ByteArrayOutputStream(); // size ==> to write the quadtree
    ObjectOutputStream oos = new ObjectOutputStream(size); //oos ==> into bytes
    oos.writeObject(root);
    oos.flush();
    oos.close();
    return size.toByteArray().length;
}
```

#### 4.1.1.16. Prosedur printTableHeader(), printRow(), dan printTableFooter()

```
public static void printTableHeader() {
    System.out.println(ANSI_CYAN + "+-----+-----+-----+");
    System.out.println(ANSI_CYAN + "|       Parameter      |       Nilai      |" + ANSI_RESET);
    System.out.println(ANSI_CYAN + "+-----+-----+-----+");
}

public static void printRow(String parameter, String value, String color) {
    System.out.printf(format:"| %-4s | %-2s |\n", color + parameter + ANSI_RESET, value);
}

public static void printTableFooter() {
    System.out.println(ANSI_CYAN + "+-----+-----+-----+");
}
```

#### 4.1.2. GifSequenceWriter.java

```
import javax.imageio.*;
import javax.imageio.metadata.*;
import javax.imageio.stream.*;
import java.awt.image.*;
import java.io.*;

public class GifSequenceWriter {
    private ImageWriter gifWriter;
    private ImageWriteParam imageWriteParam;
    private IIOImage imageMetaDate;

    public GifSequenceWriter(ImageOutputStream outputStream, int imageType, int timeBetweenFramesMS, boolean loop) throws IOException {
        gifWriter = ImageIO.getImageWritersBySuffix(fileSuffix:"gif").next();
        imageWriteParam = gifWriter.getDefaultWriteParam();

        ImageTypeSpecifier imageTypeSpecifier = ImageTypeSpecifier.createFromBufferedImageType(imageType);
        imageMetaDate = gifWriter.getDefaultImageMetadata(imageTypeSpecifier, imageWriteParam);

        String metaFormatName = imageMetaDate.getNativeMetadataFormatName();
        IIOMetadataNode root = (IIOMetadataNode) imageMetaDate.getAsTree(metaFormatName);

        IIOMetadataNode gce = new IIOMetadataNode(nodeName:"GraphicControlExtension");
        gce.setAttribute(name:"disposalMethod", value:"none");
        gce.setAttribute(name:"userInputFlag", value:"FALSE");
        gce.setAttribute(name:"transparentColorFlag", value:"FALSE");
        gce.setAttribute(name:"delayTime", Integer.toString(timeBetweenFramesMS / 10));
        gce.setAttribute(name:"transparentColorIndex", value:"0");
        root.appendChild(gce);

        IIOMetadataNode appExtensions = new IIOMetadataNode(nodeName:"ApplicationExtensions");
        IIOMetadataNode appNode = new IIOMetadataNode(nodeName:"ApplicationExtension");

        appNode.setAttribute(name:"applicationID", value:"NETSCAPE");
        appNode.setAttribute(name:"authenticationCode", value:"2.0");

        byte[] loopBytes = new byte[]{0xd, (byte) (loop ? 0 : 1), 0};
        appNode.setUserObject(loopBytes);
        appExtensions.appendChild(appNode);
        root.appendChild(appExtensions);
    }
}
```

```
    imageMetaDate.setFromTree(metaFormatName, root);
    gifWriter.setOutput(outputStream);
    gifWriter.prepareWriteSequence(streamMetadata:null);
}

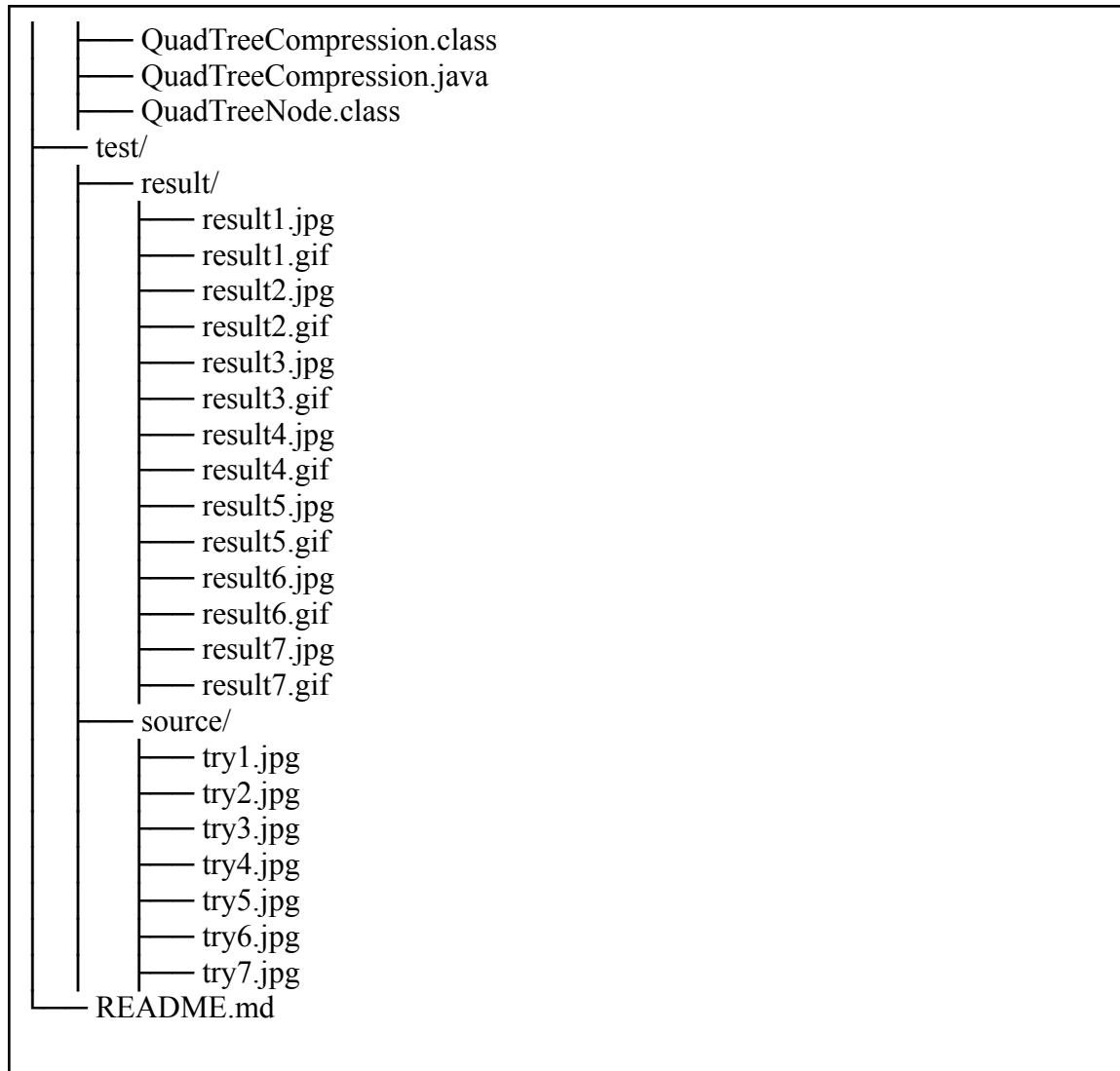
public void writeToSequence(RenderedImage img) throws IOException {
    gifWriter.writeToSequence(new IIOMImage(img, thumbnails:null, imageMetaDate), imageWriteParam);
}

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

## 4.2. Struktur Data

### 4.2.1 Struktur Data Repository

```
Tucil2_13523007_13523107/
└── bin/
└── doc/
    └── Tucil2_13523007_13523107.pdf
└── src/
    ├── GifSequenceWriter.class
    └── GifSequenceWriter.java
```



1. doc

Folder berisi dokumen spesifikasi tugas kecil 1 Strategi Algoritma dan laporan.

2. src

Folder berisi algoritma dalam kompresi gambar menggunakan Quadtree sesuai spesifikasi dengan menggunakan bahasa pemrograman Java

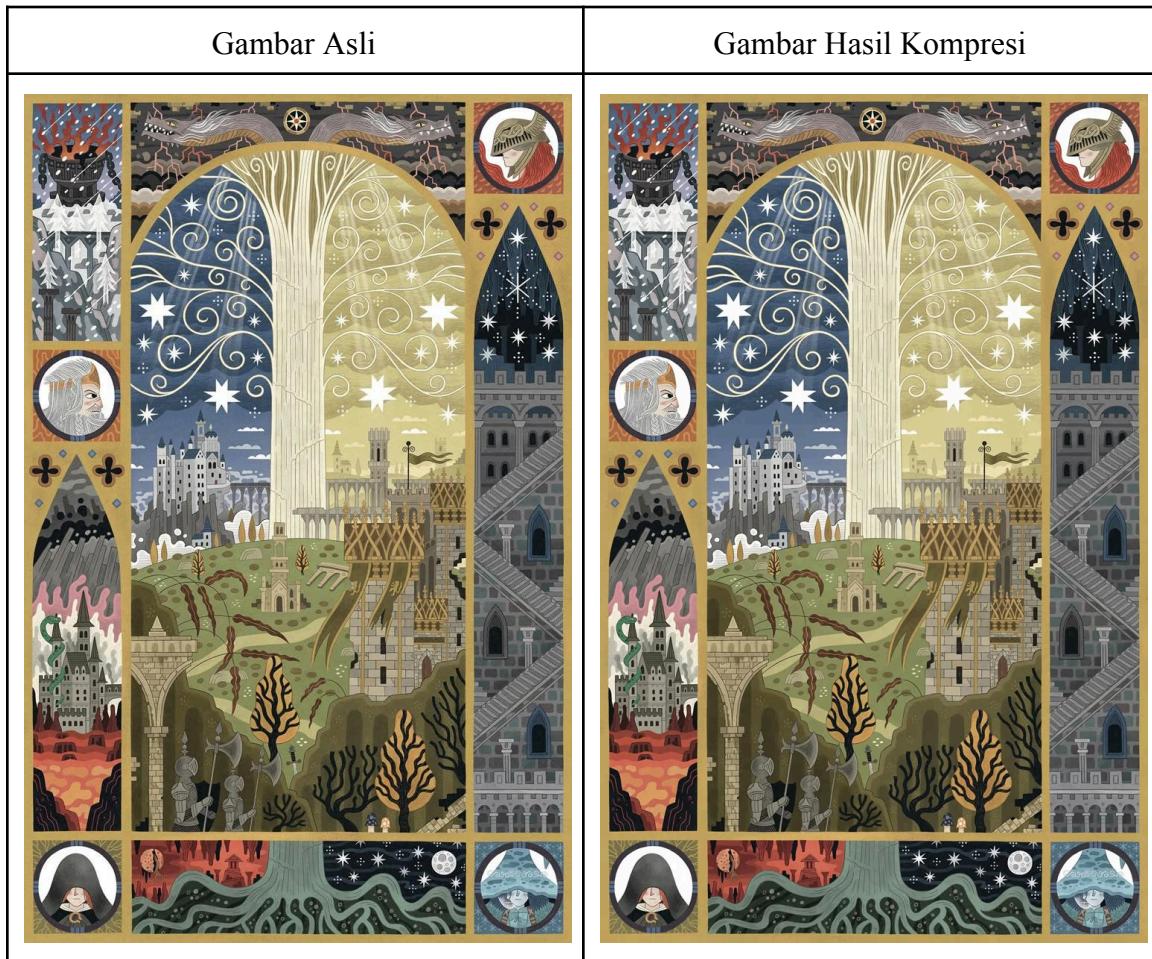
3. test

File berisi folder source yang berisi gambar test case yang akan diujikan. Folder result berisi gambar dwan gif hasil kompresi gambar dengan metode Quadtree.

## 4.3. Pengujian

### 4.3.1. Pengujian 1

Data Input
Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\source\try1.jpg
Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 1
Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0
Masukkan threshold: 10
Masukkan ukuran blok minimum: 0
Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\result\result1.jpg
Masukkan nama file GIF hasil (akhiri dengan .gif): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\result\result1.gif



Hasil pada CLI

**Hasil Kompresi:**

Parameter	Nilai
Ukuran Awal	234573 bytes
Ukuran Akhir	43278049 bytes
Kompresi	-18349,71%
Waktu Eksekusi	2860,40 ms
Kedalaman Maksimal	11
Total Simpul	737692

**4.3.2. Pengujian 2****Data Input**

Masukkan nama gambar (beserta ekstensi .jpg/.jpeg/.png):

C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\source\try2.jpg

Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 2

Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 1,0

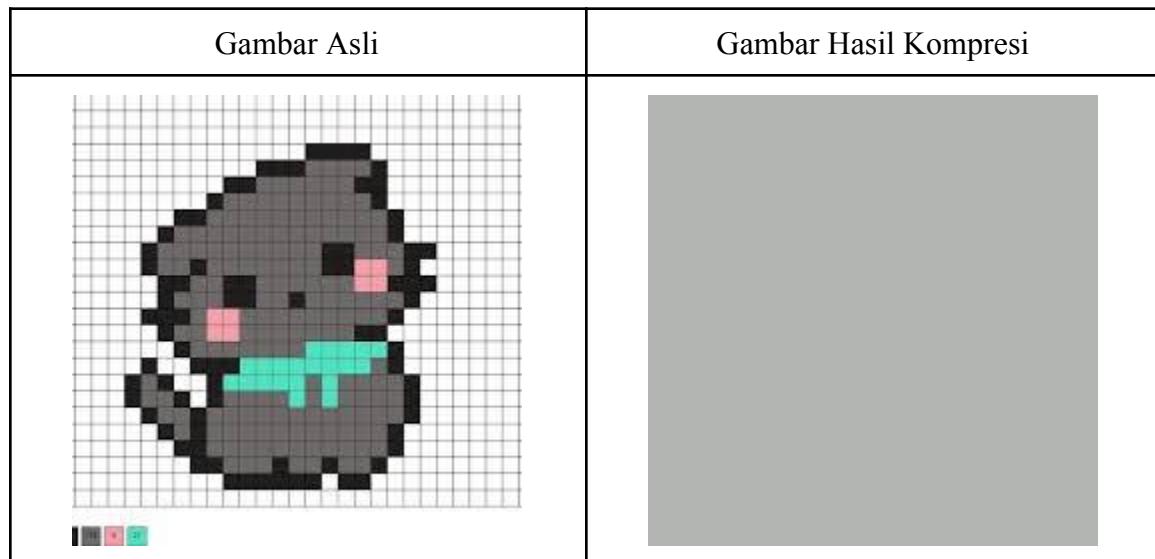
Target Compression: 1.0

Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg/.png):

C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\result\result2.jpg

Masukkan nama file GIF hasil (akhiri dengan .gif):

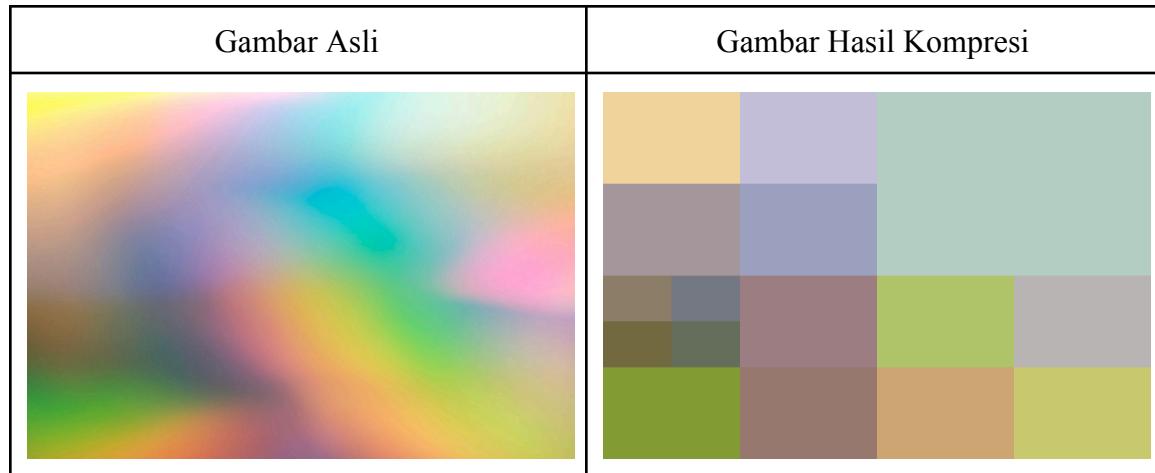
C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\result\result2.gif



Hasil pada CLI	
<b>Hasil Kompresi:</b>	
Parameter	Nilai
Ukuran Awal	10659 bytes
Ukuran Akhir	177 bytes
Kompresi	98,34%
Waktu Eksekusi	136,81 ms
Kedalaman Maksimal	0
Total Simpul	1

#### 4.3.3. Pengujian 3

Data Input
Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\source\try3.jpg
Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 3
Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0
Masukkan threshold: 150
Masukkan ukuran blok minimum: 1000
Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\result\result3.jpg
Masukkan nama file GIF hasil (akhiri dengan .gif): C:\Users\LENOVO\Downloads\Tucil2\Tucil2_13523007_13523107\test\result\result3.gif



### Hasil pada CLI

Hasil Kompresi:

Parameter	Nilai
Ukuran Awal	1137996 bytes
Ukuran Akhir	1057 bytes
Kompresi	99,91%
Waktu Eksekusi	18846,61 ms
Kedalaman Maksimal	3
Total Simpul	16

#### 4.3.4. Pengujian 4

##### Data Input

Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png): D:\STIMA\Tucil2\_13523007\_13523107\test\source\try4.png

Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 4

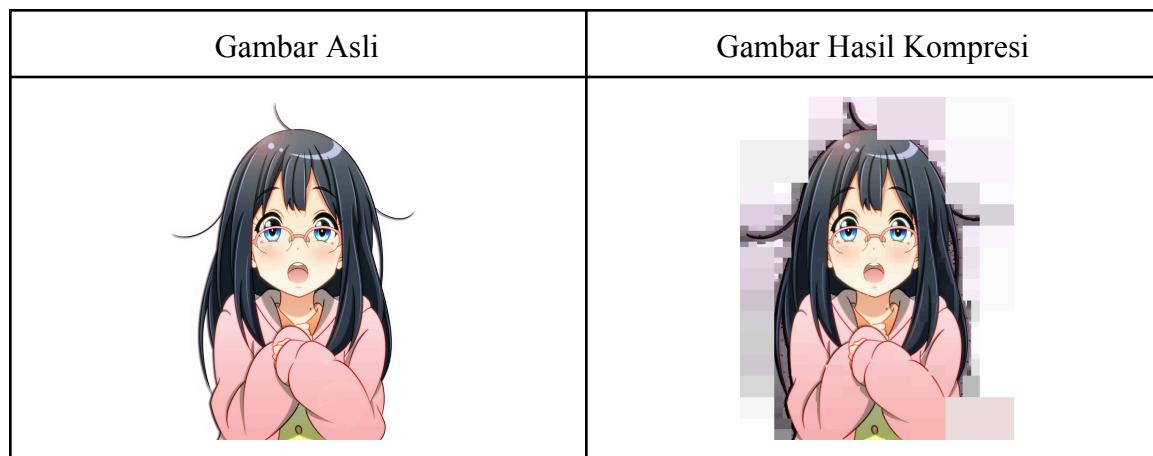
Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0

Masukkan threshold: 2,75

Masukkan ukuran blok minimum: 0

Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): D:\STIMA\Tucil2\_13523007\_13523107\test\result\result4.png

Masukkan nama file GIF hasil (akhiri dengan .gif): D:\STIMA\Tucil2\_13523007\_13523107\test\result\result4.gif



### Hasil pada CLI

```

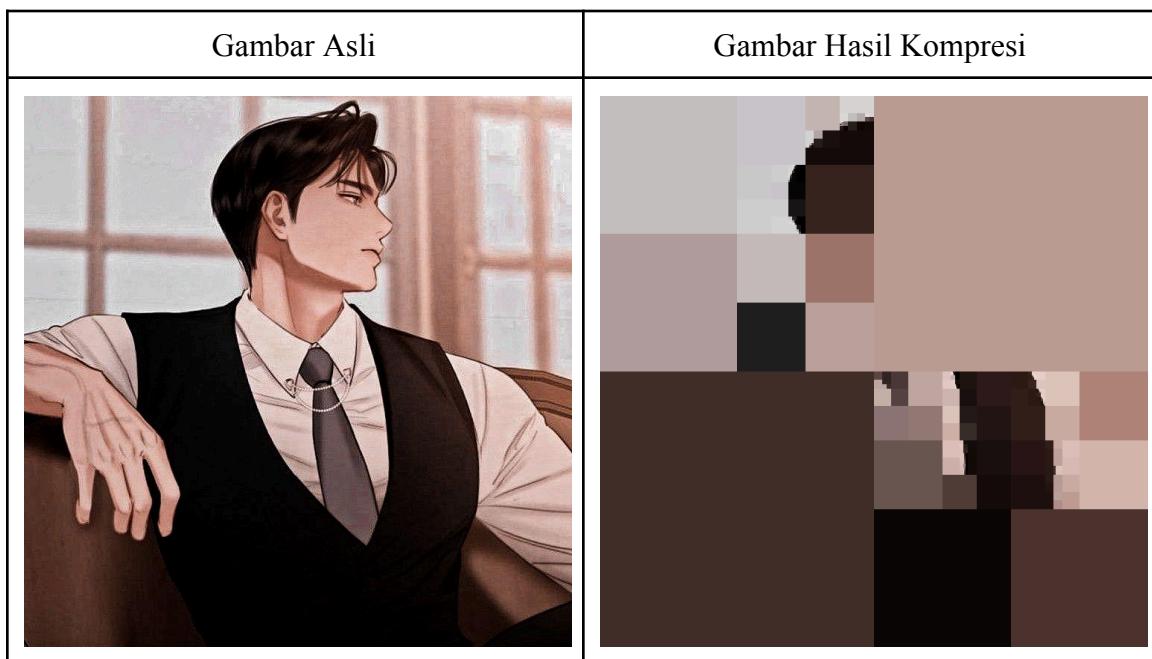
Hasil Kompresi:
+---+-----+-----+
|     Parameter      |     Nilai    |
+---+-----+-----+
|  Ukuran Awal      | 834424 bytes |
|  Ukuran Akhir     | 4493457 bytes |
|  Kompresi          | -438,51%   |
|  Waktu Eksekusi   | 6568,12 ms  |
|  Kedalaman Maksimal| 10          |
|  Total Simpul      | 76591       |
+---+-----+-----+
Gambar hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result4.png
GIF hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result4.gif

```

#### 4.3.5. Pengujian 5

##### Data Input

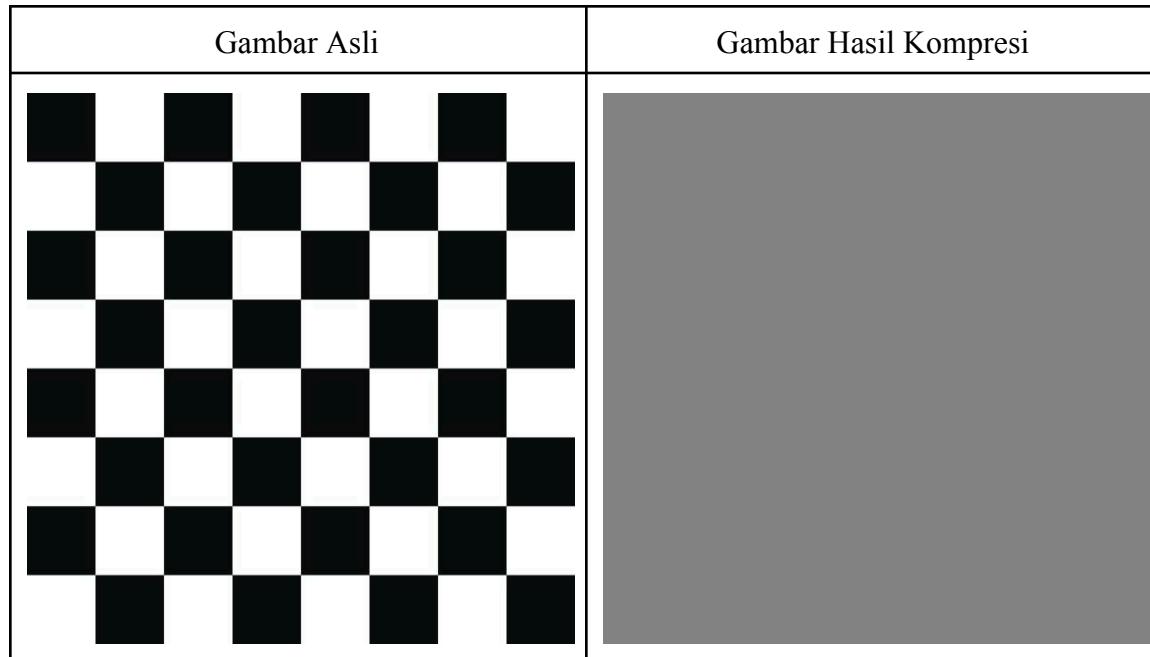
Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png): D:\STIMA\Tucil2\_13523007\_13523107\test\source\try5.jpg  
Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 5  
Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0  
Masukkan threshold: 0,9881  
Masukkan ukuran blok minimum: 0  
Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): D:\STIMA\Tucil2\_13523007\_13523107\test\result\result5.jpg  
Masukkan nama file GIF hasil (akhiri dengan .gif): D:\STIMA\Tucil2\_13523007\_13523107\test\result\result5.gif



Hasil pada CLI	
<b>Hasil Kompresi:</b>	
Parameter	Nilai
Ukuran Awal	66977 bytes
Ukuran Akhir	19009 bytes
Kompresi	71,62%
Waktu Eksekusi	1703,08 ms
Kedalaman Maksimal	10
Total Simpul	322
Gambar hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result5.jpg	
GIF hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result5.gif	

#### 4.3.6. Pengujian 6

Data Input
Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png): D:\STIMA\Tucil2_13523007_13523107\test\source\try6.png
Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 1
Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0
Masukkan threshold: 0
Masukkan ukuran blok minimum: 23000
Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png): D:\STIMA\Tucil2_13523007_13523107\test\result\result6.png
Masukkan nama file GIF hasil (akhiri dengan .gif): D:\STIMA\Tucil2_13523007_13523107\test\result\result6.gif



### Hasil pada CLI

```
Hasil Kompresi:  
+---+  
| Parameter | Nilai |  
+---+  
| Ukuran Awal | 16539 bytes |  
| Ukuran Akhir | 1057 bytes |  
| Kompresi | 93,61% |  
| Waktu Eksekusi | 622,01 ms |  
| Kedalaman Maksimal | 2 |  
| Total Simpul | 16 |  
+---+  
Gambar hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result6.jpg  
GIF hasil disimpan di: D:\STIMA\Tucil2_13523007_13523107\test\result\result6.gif
```

### 4.3.7. Pengujian 7

#### Data Input

Masukkan nama gambar (beserta ekstensi .jpg/.jpeg./.png):

C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\source\try3.jpg

Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 3

Pilih metode error (1: Variansi, 2: MAD, 3: Max Pixel Difference, 4: Entropy, 5: SSIM): 3

Masukkan target persentase kompresi (beri nilai 0 jika ingin menonaktifkan): 0

Masukkan threshold: 0

Masukkan ukuran blok minimum: 80

Masukkan nama gambar hasil (beserta ekstensinya .jpg/.jpeg./.png):

C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\result\result3.jpg

Masukkan nama file GIF hasil (akhiri dengan .gif):

C:\Users\LENOVO\Downloads\Tucil2\Tucil2\_13523007\_13523107\test\result\result3.gif

Gambar Asli

Gambar Hasil Kompresi



Hasil pada CLI

**Hasil Kompresi:**

Parameter	Nilai
Ukuran Awal	7935132 bytes
Ukuran Akhir	3844897 bytes
Kompresi	51,55%
Waktu Eksekusi	33654,63 ms
Kedalaman Maksimal	8
Total Simpul	65536

#### 4.4. Analisis Hasil Pengujian

Berdasarkan hasil pengujian, dapat disimpulkan bahwa terdapat hubungan yang erat antara nilai threshold dan ukuran minimum blok (minimum block size) dalam proses kompresi

gambar menggunakan algoritma QuadTree. Kedua parameter ini saling memengaruhi dalam menentukan tingkat presisi kompresi dan ukuran akhir file gambar. Semakin kecil nilai threshold dan ukuran minimum blok yang digunakan, maka semakin banyak simpul (node) yang dihasilkan dalam pohon, yang berarti gambar dikompres dengan lebih presisi, namun dengan konsekuensi ukuran file yang mungkin tidak jauh berkurang.

Untuk metode perhitungan error seperti Entropy dan SSIM (Structural Similarity Index), diperlukan nilai threshold yang sangat kecil untuk menghasilkan pembagian blok yang sesuai, misalnya  $\text{threshold} \leq 1$  untuk SSIM. Hal ini dikarenakan metrik tersebut cukup sensitif terhadap perubahan kecil pada data piksel.

Penggunaan fitur target kompresi menambah kompleksitas perhitungan karena membutuhkan iterasi berulang menggunakan binary search untuk menemukan threshold optimal yang mendekati target. Proses ini secara alami memakan waktu komputasi yang lebih lama dibandingkan metode kompresi langsung dengan threshold tetap.

Selain itu, dalam beberapa kasus, algoritma dapat menghasilkan tingkat kompresi yang bernilai negatif, yaitu ukuran file hasil kompresi lebih besar dari ukuran file asli. Hal ini terjadi ketika proses pembagian gambar terlalu berlebihan, menghasilkan terlalu banyak simpul yang perlu disimpan, sehingga ukuran struktur data menjadi besar. Sebab itu, penting untuk menyeimbangkan antara akurasi kompresi, ukuran file, dan waktu proses, serta menyesuaikan metode error dan parameter threshold terhadap karakteristik gambar yang dikompresi.

Program yang dikembangkan dibantu dengan pendekatan **Breadth-First Search (BFS)** untuk membangun pohon quadtree secara iteratif. Kompleksitas waktu algoritma secara umum bergantung pada ukuran gambar dan kedalaman pohon (depth). Secara umum, kompleksitas waktu algoritma Divide and Conquer dapat dinyatakan dalam bentuk:

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

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

Dalam konteks quadtree:

Fungsi  $f(n)$  merepresentasikan waktu pemrosesan satu blok gambar sebelum dibagi lebih lanjut.

Operasi yang dilakukan meliputi:

- Menghitung rata-rata warna dari blok saat ini (averageColor),
- Menghitung nilai error terhadap rata-rata warna (computeError).

Kedua operasi tersebut harus memproses seluruh piksel di dalam blok, sehingga kompleksitasnya adalah  $O(n^2)$ , dengan  $n$  adalah panjang sisi blok. Karena setiap blok dibagi menjadi empat kuadran yang diproses secara terpisah, maka terdapat 4 buah pemanggilan rekursif terhadap blok ukuran  $n/2$ , sehingga bentuk kompleksitasnya adalah:

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

Tahapan *merge* atau penggabungan hasil dari subbagian gambar tidak memerlukan proses tambahan yang signifikan, sehingga dianggap memiliki kompleksitas konstan  $O(1)$ .

Dalam kasus terburuk, pembagian kuadran dapat berlangsung hingga mencapai ukuran piksel individual, membentuk pohon quadtree dengan kedalaman sekitar  $\log_2(n)$ . Karena pada setiap level, seluruh piksel di gambar (berjumlah  $n^2$ ) tetap diperiksa melalui operasi computeError dan averageColor, maka total kompleksitas waktu algoritma dalam kasus terburuk adalah:

$$T(n) = O(n^2 \log n)$$

Namun, dalam praktiknya, kompleksitas aktual bisa lebih rendah. Hal ini karena pembagian blok dihentikan lebih awal jika nilai error sudah di bawah ambang batas (**threshold**) atau jika ukuran blok sudah mencapai batas minimum. Pemangkasan ini mengurangi kedalaman pohon dan jumlah total node yang diproses. Penggunaan BFS juga memberikan visualisasi progresif berdasarkan kedalaman dan memungkinkan pengendalian kualitas serta efisiensi proses kompresi gambar.

## **BAB 5**

### **KESIMPULAN DAN SARAN**

#### **5.1. Kesimpulan**

Dalam tugas kecil ini, kami berhasil merancang dan mengimplementasikan algoritma kompresi gambar menggunakan metode QuadTree. Implementasi ini didasarkan pada pendekatan divide and conquer, di mana proses kompresi dilakukan secara efisien dengan membagi citra menjadi bagian-bagian yang lebih kecil secara rekursif hingga memenuhi batasan threshold error tertentu dan ukuran minimum blok. Pendekatan ini memungkinkan keseimbangan antara kualitas gambar dan efisiensi ukuran file hasil kompresi.

Algoritma ini juga dirancang dengan fleksibilitas dalam hal pemilihan metode error, seperti Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, Entropy, dan Structural Similarity Index (SSIM). Hal ini memungkinkan algoritma untuk disesuaikan dengan jenis gambar atau kebutuhan spesifik pengguna, baik untuk mempertahankan detail struktur gambar maupun untuk efisiensi ukuran.

Selain itu, untuk mencapai target kompresi yang diinginkan secara otomatis, kami mengimplementasikan strategi pencarian threshold optimal menggunakan binary search. Strategi ini memastikan bahwa algoritma mampu menemukan nilai ambang batas error yang paling tepat untuk mendekati rasio kompresi yang ditetapkan, tanpa perlu pengaturan manual yang berulang.

Sebagai tambahan dari fitur utama, kami juga mengimplementasikan fitur bonus berupa pembuatan GIF animasi yang menampilkan proses pembentukan blok QuadTree secara visual. Fitur ini berguna untuk membantu pemahaman terhadap cara kerja algoritma, serta menunjukkan proses kompresi secara interaktif.

Secara keseluruhan, algoritma yang kami bangun menunjukkan bahwa pendekatan QuadTree berbasis divide and conquer sangat efektif untuk kompresi gambar spasial, serta memberikan kontrol yang tinggi terhadap kualitas dan ukuran hasil kompresi.

#### **5.2. Saran**

Dari Tugas Kecil 2 IF2211 Strategi Algoritma ini, kami menyadari pentingnya ketersediaan edge case dan test case yang beragam agar dapat mengevaluasi apakah algoritma yang telah dirancang bekerja secara optimal dalam berbagai kondisi. Kami juga menyarankan

untuk selalu bersikap teliti dalam merancang algoritma, serta mempertimbangkan berbagai aspek penting, seperti penentuan batas awal threshold pada proses pencapaian target kompresi.

## 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 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	✓	

### Tautan Repository GitHub

[https://github.com/mineraleee/Tucil2\\_13523007\\_13523107](https://github.com/mineraleee/Tucil2_13523007_13523107)

## DAFTAR PUSTAKA

- Munir, R. 2025. “Algoritma Divide and Conquer (Bagian 1)”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf). [1 April 2025].
- Munir, R. 2025. “Algoritma Divide and Conquer (Bagian 2)”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf). [1 April 2025].
- Munir, R. 2025. “Algoritma Divide and Conquer (Bagian 3)”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-\(2025\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf). [1 April 2025].
- Munir, R. 2025. “Algoritma Divide and Conquer (Bagian 4)”. [Online]. Tersedia: [https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-\(2025\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian4.pdf). [1 April 2025].
- Sundararaju, M. 2024. “Java Range Search”. [Online]. Tersedia: <https://www.baeldung.com/java-range-search>. Ditinjau oleh: Kevin Gilmore. [Diakses: 7 April 2025].
- Murat Arat, M. 2020. “RGB to Grayscale Conversion Formulas”. [Online]. Tersedia: [https://mmuratarat.github.io/2020-05-13/rgb\\_to\\_grayscale\\_formulas](https://mmuratarat.github.io/2020-05-13/rgb_to_grayscale_formulas). [Diakses: 11 April 2025].
- Java2s. “AnimatedGifEncoder”. [Online]. Tersedia: <http://www.java2s.com/Code/Java/2D-Graphics-GUI/AnimatedGifEncoder.htm>. [Diakses: 11 April 2025].