

IF2211 Strategi Algoritma  
**Kompresi Gambar Dengan Metode Quadtree**

**Laporan Tugas Kecil 2**

Disusun untuk memenuhi tugas mata kuliah Strategi Algoritma pada Semester 4  
(empat) Tahun Akademik 2024/2025



Disusun Oleh:  
**Brian Ricardo Tamin (13523126)**

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

**BANDUNG 2025**

# DAFTAR ISI

<b>DAFTAR ISI.....</b>	<b>2</b>
<b>BAB I DESKRIPSI MASALAH.....</b>	<b>3</b>
<b>BAB II SPESIFIKASI TUGAS.....</b>	<b>4</b>
2.1. Input.....	4
2.2. Output.....	4
<b>BAB III TEORI SINGKAT.....</b>	<b>5</b>
<b>BAB IV SOURCE CODE.....</b>	<b>6</b>
4.1 Library.....	6
4.2 Struktur.....	6
4.3 Struktur Data.....	7
4.3.1 Struktur Data QuadTree.....	7
4.3.2 Struktur Data Point.....	8
4.3.3 Struktur Data RGB.....	8
4.4 Fungsi & Prosedur.....	9
4.4.1 ProcessImage.java.....	9
4.4.2 ErrorMeasurement.java.....	9
4.4.3 GifWriter.java.....	10
4.5 Analisis Algoritma.....	11
4.5.1 Constraints.....	11
4.5.2 Cara Kerja Algoritma.....	11
4.5.3 Kompleksitas Algoritma.....	14
4.6 Implementasi Bonus.....	15
4.6.1 Implementasi Structural Similarity Index (SSIM).....	15
4.6.2 Implementasi Target Kompresi (%).....	16
4.6.1 Implementasi GIF.....	18
<b>BAB VI TEST CASE.....</b>	<b>20</b>
<b>BAB VII KESIMPULAN &amp; SARAN.....</b>	<b>30</b>
7.1 Kesimpulan.....	30
7.2 Saran.....	30
<b>BAB VIII DAFTAR PUSTAKA.....</b>	<b>31</b>
<b>LAMPIRAN.....</b>	<b>32</b>

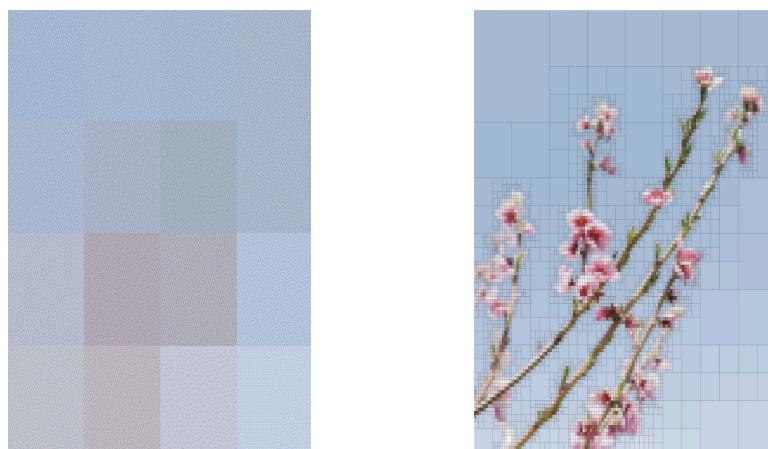
# BAB I

## DESKRIPSI MASALAH

Perkembangan teknologi digital saat ini telah menyebabkan pertumbuhan eksponensial jumlah data gambar yang disimpan dan dipertukarkan. Gambar digital, terutama yang beresolusi tinggi, memiliki ukuran file yang besar sehingga membutuhkan penyimpanan dan transmisi yang efisien. Salah satu solusi untuk mengatasi masalah tersebut adalah dengan melakukan kompresi gambar. Namun, kompresi gambar harus mampu mengurangi ukuran file tanpa mengorbankan kualitas visual secara signifikan.

Dalam konteks ini, metode Quadtree muncul sebagai salah satu teknik yang menjanjikan. Metode ini memanfaatkan pendekatan divide-and-conquer dengan cara membagi gambar menjadi empat bagian secara rekursif berdasarkan tingkat keseragaman nilai piksel (*intensity value*). Jika suatu blok dianggap homogen berdasarkan analisis nilai intensitas dari masing-masing kanal warna, maka block tersebut dapat disimpan tanpa perlu dibagi lagi. Sebaliknya, blok yang memiliki variasi yang tinggi akan dibagi lebih lanjut untuk menangkap detail-detail yang penting.

Fokus utama dalam pengembangan algoritma ini adalah menentukan kriteria pembagian blok secara optimal. Hal ini melibatkan pengukuran error dalam tiap blok, yang dapat dilakukan dengan berbagai metode seperti variansi, *Mean Absolute Deviation (MAD)*, perbedaan maksimum antar piksel, dan entropi. Selain itu, terdapat pula pendekatan lanjutan dengan menggunakan *Structural Similarity Index (SSIM)* untuk menilai kesamaan struktur antara blok gambar sebelum dan sesudah pembagian. Dengan demikian, pemilihan metode pengukuran error yang tepat dan penentuan ambang batas (*threshold*) yang sesuai menjadi kunci agar kompresi dapat mengurangi ukuran file sekaligus mempertahankan kualitas gambar.



**Gambar 1.** Kompresi Gambar Menggunakan Quadtree  
(Sumber : <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

## **BAB II**

### **SPESIFIKASI TUGAS**

Tugas ini bertujuan untuk mengimplementasikan algoritma kompresi gambar berbasis Quadtree dengan metode divide and conquer. Gambar yang dimasukkan akan diproses dalam format matriks piksel dengan representasi 24-bit RGB (8-bit per kanal), yang dibagi menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Program ini harus mampu melakukan kompresi gambar dengan mengukur error antara blok gambar menggunakan beberapa metode, seperti variansi, *Mean Absolute Deviation (MAD)*, perbedaan nilai piksel maksimum, entropi, dan *Structural Similarity Index (SSIM)*. Selain itu, program juga memungkinkan pengguna untuk menentukan nilai ambang batas (*threshold*) untuk error dan ukuran blok minimum, serta target persentase kompresi untuk mengatur efisiensi kompresi.

Setelah proses kompresi, gambar yang telah diproses akan disimpan dan visualisasi proses kompresi dapat ditampilkan dalam bentuk GIF sebagai bonus. Program ini harus dapat dijalankan pada platform Windows dan Linux, serta menyediakan laporan yang mencakup statistik dari struktur Quadtree yang dihasilkan, termasuk kedalaman pohon dan jumlah simpul. Selain itu, analisis kompleksitas algoritma dan hasil eksperimen juga harus disertakan dalam laporan.

#### **2.1. Input**

- Alamat absolut gambar yang akan dikompresi
- Metode perhitungan error
- Ambang batas (*threshold*)
- Ukuran blok minimum
- Target persentase kompresi
- Alamat absolut gambar hasil kompresi
- Alamat absolut gif (bonus)

#### **2.2. Output**

- Waktu eksekusi
- Ukuran gambar sebelum
- Ukuran gambar setelah
- Persentase kompresi
- Kedalaman pohon
- Banyak simpul pada pohon
- Gambar hasil kompresi pada alamat yang sudah ditentukan
- Gif proses kompresi pada alamat yang sudah ditentukan (bonus)

## BAB III

### TEORI SINGKAT

Tugas Kecil 2 ini mengimplementasikan konsep kompresi gambar menggunakan metode Quadtree, yang merupakan salah satu teknik pengolahan gambar berbasis struktur data hierarkis. Konsep dasar Quadtree adalah membagi ruang atau gambar menjadi bagian-bagian yang lebih kecil berdasarkan keseragaman warna atau intensitas piksel. Proses kompresi dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa keseragaman setiap bagian tersebut dengan membandingkan nilai warna RGB (merah, hijau, dan biru) dari piksel-piksel di dalamnya. Jika bagian gambar tersebut tidak seragam, maka bagian tersebut akan dibagi lebih lanjut, dan proses ini akan terus berlangsung hingga mencapai tingkat keseragaman tertentu.

Quadtree memungkinkan pengolahan gambar secara efisien karena menghilangkan redundansi yang ada di bagian-bagian gambar yang seragam. Dalam hal ini, Quadtree berfungsi sebagai struktur data yang menyimpan informasi mengenai blok gambar yang telah diproses, dimana setiap simpul dalam pohon Quadtree merepresentasikan bagian gambar dengan ukuran dan warna rata-rata tertentu. Struktur ini memanfaatkan sifat hierarkis untuk mengurangi ukuran data gambar dengan cara menggantikan bagian-bagian gambar yang seragam dengan satu representasi rata-rata warna.

Penerapan Quadtree dalam kompresi gambar juga menggabungkan metode pengukuran error untuk menentukan apakah sebuah blok gambar perlu dibagi lagi atau tidak. Metode-metode seperti variansi, *Entropy*, *Mean Absolute Deviation (MAD)*, dan *Max Pixel Difference* digunakan untuk mengukur perbedaan antara nilai-nilai piksel dalam blok tersebut. Jika perbedaan antara piksel-piksel di dalam blok lebih besar dari ambang batas yang ditentukan, maka blok tersebut akan dibagi lagi untuk memperoleh kompresi yang lebih optimal. Dengan menggunakan pendekatan ini, Quadtree dapat menyesuaikan tingkat kompresi berdasarkan kebutuhan dan parameter yang ditentukan oleh pengguna. Terdapat *constraint* lain juga dalam menghentikan pembagian yaitu minimum luasan blok suatu node Quadtree.

Selain itu, konsep Quadtree dalam kompresi gambar juga dapat dikombinasikan dengan konsep Structural Similarity Index (SSIM) untuk mengukur kualitas gambar setelah kompresi. *SSIM* berfungsi untuk membandingkan kesamaan struktur antara gambar yang asli dan gambar hasil kompresi, memberikan penilaian lebih mendalam tentang kualitas visual gambar yang telah diproses. Konsep-konsep ini menggabungkan teori pengolahan gambar, kompresi data, dan pengukuran kualitas untuk menghasilkan solusi kompresi gambar yang efisien dan berkualitas tinggi.

# BAB IV

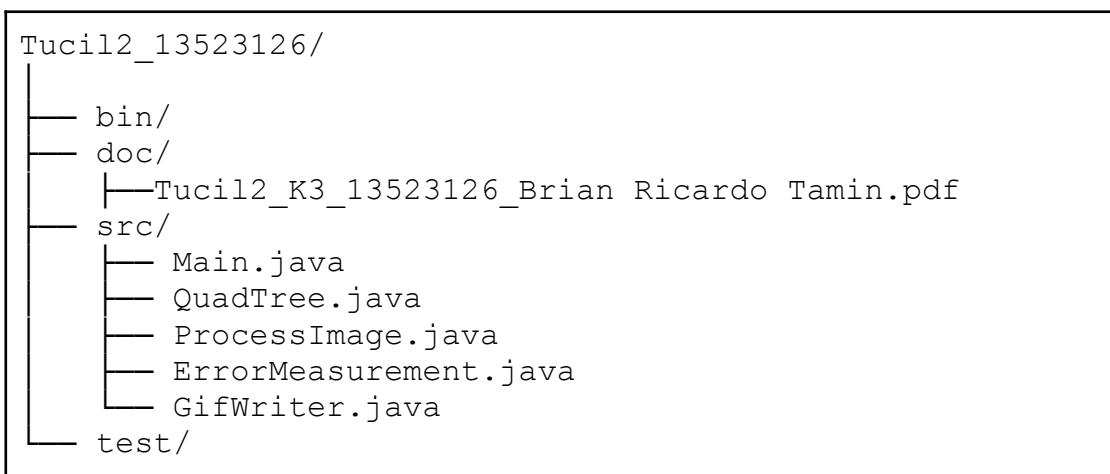
## SOURCE CODE

Tucil ini dikembangkan menggunakan Java (JDK 17+) untuk mengelola source code pada CLI hingga dapat mengoutput hasil dari proses divide and conquer sesuai ekspektasi.

### 4.1 Library

- *java.util.ArrayList* : Menyimpan daftar frame gambar untuk animasi GIF.
- *java.util.List* : Mendefinisikan jenis koleksi untuk menyimpan frame GIF.
- *java.util.Scanner* : Menerima input dari pengguna.
- *javax.imageio.stream.FileImageOutputStream* : Menulis file gambar ke disk.
- *java.io.File* : Mengecek keberadaan direktori file.
- *java.io.IOException* : Menangani kesalahan I/O yang mungkin terjadi.
- *java.awt.image BufferedImage* : Mewakili gambar dalam memori.
- *java.io.ByteArrayOutputStream* : Menulis data ke dalam array byte.
- *javax.imageio.ImageIO* : Membaca dan menyimpan gambar.
- *java.awt.Color* : Mengelola warna gambar.
- *java.imageio.ImageWriteParam* : Menyediakan pengaturan proses kompresi.
- *javax.imageio.ImageWriter* : Menulis gambar dalam format tertentu.
- *javafx.imageio.ImageTypeSpecifier* : Menentukan jenis gambar saat menulis.
- *javax.imageio.metadata.IIOMetadata* : Menambahkan metadata pada gambar.
- *javax.imageio.stream.ImageOutputStream* : output gambar dalam format GIF.
- *java.awt.image.RenderedImage* : Mendefinisikan tipe gambar.

### 4.2 Struktur



- *Main.java* : Program inti mencakup keseluruhan proses.
- *QuadTree.java* : Program yang berisikan struktur data hierarkis quadtree.
- *ProcessImage.java* : Program untuk mengolah image file.
- *ErrorMeasurement.java* : Program berisikan pengukuran value error.
- *GifWriter.java* : Program untuk mengurus gif pada quadtree (bonus).

## 4.3 Struktur Data

### 4.3.1 Struktur Data QuadTree

Atribut/Metode	Tipe	Penjelasan Singkat
Point	class	Menyimpan koordinat titik (x,y).
RGB	class	Menyimpan nilai intensitas warna merah, hijau, dan biru dari piksel.
point	Point	Menyimpan titik pusat dari area yang sedang diproses.
width	integer	Lebar area dalam pohon QuadTree.
height	integer	Tinggi area dalam pohon QuadTree.
intensityValue	RGB	Menyimpan rata-rata nilai intensitas warna (RGB) dalam area yang diwakili oleh node ini.
northEast, northWest, southEast, southWest	QuadTree	Referensi keempat anak dari node ini (subbagian dari area).
QuadTree(int w, int h)	constructor	Konstruktor untuk inisialisasi pohon QuadTree dengan lebar dan tinggi yang diberikan.
setWidth(int width)	setter	Menetapkan nilai lebar area.
setHeight(int height)	setter	Menetapkan nilai tinggi area.
setPoint(int x, int y)	setter	Menetapkan titik pusat (x,y).
getRedIntensityValue()	getter	Mengembalikan nilai rata-rata intensitas warna merah dari area.
getGreenIntensityValue()	getter	Mengembalikan nilai rata-rata intensitas warna hijau dari area.
getBlueIntensityValue()	getter	Mengembalikan nilai rata-rata intensitas warna biru dari area.
setIntensityValue()	method	Menghitung dan menetapkan rata-rata nilai intensitas warna (RGB) untuk area yang diwakili

		oleh node ini.
isLeaf()	method	Mengembalikan true jika node ini adalah daun (tidak memiliki anak), false jika node ini memiliki anak.
compressQuadTree(int inputMode, double threshold, int blockSizeConstrain)	method	Metode rekursif untuk mengompresi QuadTree berdasarkan metode pengukuran kesalahan dan threshold yang ditentukan.
reconstructQuadTree(BufferedImage output)	method	Metode rekursif untuk merekonstruksi gambar dari QuadTree yang sudah terkompresi.
getDepth()	method	Mengembalikan kedalaman maksimal dari QuadTree.
getNodes()	method	Mengembalikan jumlah total node dalam QuadTree, termasuk node daun dan anak-anaknya.

#### 4.3.2 Struktur Data Point

Atribut/Metode	Tipe	Penjelasan Singkat
x	integer	Menyimpan absis titik tengah.
y	integer	Menyimpan ordinat titik tengah.
Point(int x, int y)	constructor	Konstruktor pembuatan tipe Point.

#### 4.3.3 Struktur Data RGB

Atribut/Metode	Tipe	Penjelasan Singkat
red	integer	Menyimpan intensity value merah.
green	integer	Menyimpan intensity value hijau.
blue	integer	Menyimpan intensity value biru.
RGB(int r, int g, int b)	constructor	Konstruktor pembuatan tipe RGB.

## 4.4 Fungsi & Prosedur

### 4.4.1 ProcessImage.java

Prosedur / Fungsi	Penjelasan Singkat
ProcessInputImage(String in_file)	Fungsi ini digunakan untuk memproses gambar input dari file yang diberikan. Gambar dibaca dan disimpan dalam Main.imageFile jika file valid dan memiliki ekstensi gambar yang sesuai.
isValidImageExtension(File file)	Fungsi untuk memeriksa apakah file gambar memiliki ekstensi yang valid seperti .jpg, .jpeg, atau .png. Mengembalikan true jika valid, false jika tidak.
saveImage(BufferedImage image, String outputPath)	Fungsi ini menyimpan gambar yang diberikan ke lokasi file output yang ditentukan, dengan format gambar yang sesuai (.png, .jpg, .jpeg). Mengembalikan true jika berhasil.
getImageSize(BufferedImage image, String outputPath)	Fungsi ini mengembalikan ukuran gambar yang telah di encode (dalam byte) dengan format yang sesuai , yang dihitung menggunakan ByteArrayOutputStream.

### 4.4.2 ErrorMeasurement.java

Prosedur / Fungsi	Penjelasan Singkat
specificVariance(QuadTree qt)	Fungsi ini menghitung varians untuk setiap channel warna (merah, hijau, biru) dalam area yang diwakili oleh QuadTree.
variance(QuadTree qt)	Fungsi ini mengembalikan nilai varians total (merah, hijau, biru) untuk QuadTree. Menggunakan fungsi specificVariance.
meanAbsoluteDeviation(QuadTree qt)	Fungsi ini menghitung Mean Absolute Deviation (MAD), yaitu rata-rata deviasi absolut nilai intensitas warna dalam QuadTree.

maxPixelDifference(QuadTree qt)	Fungsi ini menghitung selisih maksimum antara nilai warna piksel di dalam QuadTree untuk setiap channel warna.
entropy(QuadTree qt)	Fungsi ini menghitung entropi gambar berdasarkan distribusi intensitas warna di dalam QuadTree, menggambarkan keragaman informasi.
maenFourValue(double a, double b, double c, double d)	Fungsi ini menghitung rata-rata dari empat nilai yang diberikan.
covariance(QuadTree parent, QuadTree child1, QuadTree child2, QuadTree child3, QuadTree child4)	Fungsi ini menghitung kovarians antara node parent dan empat child node berdasarkan intensitas warna.
structuralSimilarityIndex(QuadTree parent, QuadTree child1, QuadTree child2, QuadTree child3, QuadTree child4)	Fungsi ini menghitung <i>Structural Similarity Index (SSIM)</i> antara node parent dan 4 node child untuk mengatur kemiripan struktural. Pada fungsi ini juga ditetapkan konstanta c1 dan c2 dengan value $c1 = (0.01 * 255)^2$ $c2 = (0.03 * 255)^2$ berdasarkan paper research pada <a href="#">[2]</a> dan konstanta red green blue SSIM sebesar 0.2989 (red), 0.5785 (green), dan 0.114 (blue) dari paper research pada <a href="#">[1]</a> .
errorValue(QuadTree qt, int input)	Fungsi ini mengembalikan nilai kesalahan berdasarkan mode pengukuran yang dipilih: varians, MAD, perbedaan piksel maksimum, entropi , atau SSIM.

#### 4.4.3 GifWriter.java

Prosedur / Fungsi	Penjelasan Singkat
GifWriter(ImageOutputStream outputStream, int imageType, int delay, boolean loops)	Konstruktor yang menginisialisasi objek GifWriter untuk menulis gambar dalam format GIF, dengan pengaturan seperti jenis gambar, delay antar frame, dan apakah animasi akan loop.
writeToSequence(RenderedImage img)	Fungsi ini menulis gambar (frame) ke dalam urutan GIF yang sedang

	dibangun.
close()	Menutup dan menyelesaikan penulisan urutan GIF setelah semua frame ditambahkan.

## 4.5 Analisis Algoritma

Algoritma yang digunakan pada tugas kecil kali ini adalah *divide and conquer* dimana dengan memanfaatkan struktur data quadtree, algoritma ini bekerja dengan sangat efektif dalam melakukan kompresi ukuran file. Terdapat 2 *core function* dalam peranannya untuk mengimplementasikan algoritma ini yaitu compressQuadTree() untuk mengubah informasi rgb tiap pixel image file menjadi representasi quadtree dan reconstructQuadTree untuk mengubah QuadTree yang telah dikompresi menjadi image file yang sudah dikompresi dengan persentase kompresi tertentu.

### 4.5.1 Constraints

- *Threshold* : Batas toleransi kesalahan dalam kompresi gambar (*error value*).
- *Block Size* : Ukuran blok minimum yang digunakan dalam kompresi.
- *Compression iteration limit* : Jumlah iterasi kompresi 30 iterasi. (Bonus)
- *Error method* : Setiap metode memiliki nilai rentang ideal yang berbeda.
- *Delta change* : Perubahan nilai threshold selama iterasi. (Bonus)
- *Expected output size* : Ukuran file gambar yang diharapkan setelah kompresi. (Bonus)

### 4.5.2 Cara Kerja Algoritma

Fungsi compressQuadTree()

```

Procedure compressQuadTree(inputMode, threshold, blockSizeConstrain):
    If width < 2 or height < 2 then:
        Return from procedure
    End If

    base_axis ← GetAxis()
    base_ordinate ← GetOrdinate()

    floor_width ← Floor(Width / 2)
    ceil_width ← Ceil(Width / 2)
    ceil_height ← Ceil(Height / 2)
    floor_height ← Floor(Height / 2)

    south_ordinate ← base_ordinate + Ceil(0.5 * floor_height)
    north_ordinate ← base_ordinate - Floor(0.5 * ceil_height)
    west_axis ← base_axis - Ceil(0.5 * floor_width)
    east_axis ← base_axis + Floor(0.5 * ceil_width)

    nw ← CreateQuadTree(floor_width, ceil_height)
    nw.SetPoint(west_axis, north_ordinate)
    nw.SetIntensityValue()

```

```

        ne ← CreateQuadTree(ceil_width, ceil_height)
        ne.SetPoint(east_axis, north_ordinate)
        ne.SetIntensityValue()

        se ← CreateQuadTree(ceil_width, floor_height)
        se.SetPoint(east_axis, south_ordinate)
        se.SetIntensityValue()

        sw ← CreateQuadTree(floor_width, floor_height)
        sw.SetPoint(west_axis, south_ordinate)
        sw.SetIntensityValue()

        SetChild(nw, ne, se, sw)

        If inputMode != 5 then:
            If nw.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(nw, inputMode) > threshold then:
                nw.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If ne.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(ne, inputMode) > threshold then:
                ne.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If se.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(se, inputMode) > threshold then:
                se.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If sw.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(sw, inputMode) > threshold then:
                sw.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
        Else:
            If nw.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(this, inputMode) > threshold then:
                nw.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If ne.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(this, inputMode) > threshold then:
                ne.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If se.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(this, inputMode) > threshold then:
                se.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
            If sw.GetArea() > blockSizeConstrain and
            ErrorMeasurement.ErrorValue(this, inputMode) > threshold then:
                sw.compressQuadTree(inputMode, threshold, blockSizeConstrain)
            End If
        End If
    End Procedure

```

Fungsi ini digunakan untuk melakukan kompresi gambar dengan metode QuadTree menggunakan algoritma *divide and conquer*, di mana gambar dibagi menjadi blok-blok yang lebih kecil dan dievaluasi berdasarkan ukuran kesalahan. Berikut merupakan langkah-langkah cara kerjanya:

1. Jika lebar (width) dan tinggi (height) gambar lebih kecil dari 2, maka kompresi dihentikan, dan fungsi keluar dari rekursi.
2. Jika gambar lebih besar dari ukuran batas minimum, gambar akan dibagi menjadi empat bagian (nw, ne, se, sw).

3. koordinat untuk masing-masing bagian dihitung dengan memecah gambar menjadi dua bagian (membagi lebar dan tinggi menjadi dua).
4. Untuk setiap bagian (nw, ne, se, sw), dibuat objek QuadTree baru yang berukuran sesuai dengan bagian yang telah dihitung.
5. Masing-masing objek QuadTree diberikan nilai intensitas warna dan diatur pada koordinat yang tepat.
6. Setelah membuat bagian-bagian kecil, fungsi akan memanggil dirinya sendiri untuk mengkompresi setiap bagian, dengan memeriksa apakah kesalahan (error) dari setiap bagian lebih besar dari batasan threshold.
7. Fungsi ini juga mengecek jika area blok lebih besar dari batasan ukuran blok minimum, dan jika nilai kesalahan dari blok tersebut lebih besar dari threshold.
8. Jika mode input adalah 5 (SSIM), kompresi dilakukan dengan membandingkan kesalahan total gambar sebelum dan sesudah kompresi, bukan hanya kesalahan masing-masing blok.
9. Fungsi berlanjut untuk mengevaluasi blok-blok lainnya, memperkecil atau memperbesar nilai threshold berdasarkan ukuran gambar yang dihasilkan dari setiap iterasi.

### Fungsi reconstructQuadTree()

```

Procedure reconstructQuadTree(output):
  If isLeaf() is true then:
    For y from startY() to endY() do:
      For x from startX() to endX() do:
        newColor ← new Color(getRedIntensityValue(),
        getGreenIntensityValue(), getBlueIntensityValue())
        output.SetRGB(x, y, newColor.GetRGB())
      End For
    End For
    Return from procedure
  End If

  If northEast is not null then:
    northEast.reconstructQuadTree(output)
  End If
  If northWest is not null then:
    northWest.reconstructQuadTree(output)
  End If
  If southEast is not null then:
    southEast.reconstructQuadTree(output)
  End If
  If southWest is not null then:
    southWest.reconstructQuadTree(output)
  End If
End Procedure

```

Berbeda dengan fungsi compressQuadTree(), fungsi ini digunakan untuk merekonstruksi gambar dari data QuadTree yang telah dikompresi. Berikut adalah langkah-langkah cara kerjanya:

1. Jika QuadTree saat ini adalah daun (leaf node), maka proses rekonstruksi dimulai dengan mengisi bagian gambar yang sesuai dengan nilai intensitas yang ada pada node tersebut.
2. Untuk setiap pixel dalam rentang koordinat yang ditentukan oleh node, warna dihitung berdasarkan nilai intensitas merah, hijau, dan biru yang disimpan di dalam node.
3. Nilai intensitas yang disimpan dalam node digunakan untuk membuat warna baru yang kemudian dituangkan ke setiap pixel di bagian gambar tersebut.
4. Warna baru ini dituangkan pada gambar yang diberikan sebagai parameter output menggunakan setRGB().
5. Jika node saat ini bukan daun (bukan leaf), maka proses rekonstruksi dilanjutkan ke sub-node (ne, nw, se, sw).
6. Jika sub-node tersebut tidak null, maka fungsi reconstructQuadTree dipanggil kembali untuk sub-node tersebut.
7. Proses rekonstruksi ini terjadi secara rekursif untuk semua sub-node hingga gambar sepenuhnya direkonstruksi.

#### 4.5.3 Kompleksitas Algoritma

Pada fungsi compressQuadTree(), diimplementasikan algoritma inti untuk menyelesaikan permasalahan kompresi gambar yaitu *divide and conquer*. Fungsi ini memiliki basis rekursi , yaitu berhenti jika lebar (width) atau tinggi (height) node kurang dari 2 piksel, karena blok tidak dapat dibagi lagi. Subdivisi juga dilakukan dengan melakukan pembagian ke 4 ruang (nw, ne, se, sw) dengan titik tengah menjadi acuannya.

Pada fungsi ini dimisalkan bahwa  $P$  = total jumlah piksel pada gambar. Pada setiap node, perhitungan intensitas dilakukan dengan iterasi pada seluruh piksel dalam blok, sehingga memerlukan waktu  $O(\text{node\_area})$ . Karena setiap piksel ikut diproses dan setiap level mencakup keseluruhan gambar , dengan kira-kira sebanding dengan :

$$O(P)$$

Pada kondisi ideal (pembagian yang seimbang), kedalaman tree adalah :

$$O(\log P)$$

Sehingga, total kompleksitas waktu fungsi compressQuadTree adalah :

$$O(P) \times O(\log P) = O(P \log P)$$

## 4.6 Implementasi Bonus

Pada tugas kecil kali ini, saya berhasil mengimplementasikan seluruh spesifikasi bonus dari pengimplementasian error value *Structural Similarity Index (SSIM)*, pembuatan GIF, hingga kompresi gambar sesuai input.

### 4.6.1 Implementasi *Structural Similarity Index (SSIM)*

```
Procedure structuralSimilarityIndex(parent, child1, child2, child3, child4):

    c1 ← (0.01 * 255)2
    c2 ← (0.03 * 255)2

    meanRedChild ← meanFourValue(child1.getRedIntensityValue(),
    child2.getRedIntensityValue(), child3.getRedIntensityValue(),
    child4.getRedIntensityValue())
    meanGreenChild ← meanFourValue(child1.getGreenIntensityValue(),
    child2.getGreenIntensityValue(), child3.getGreenIntensityValue(),
    child4.getGreenIntensityValue())
    meanBlueChild ← meanFourValue(child1.getBlueIntensityValue(),
    child2.getBlueIntensityValue(), child3.getBlueIntensityValue(),
    child4.getBlueIntensityValue())

    meanRedParent ← parent.getRedIntensityValue()
    meanGreenParent ← parent.getGreenIntensityValue()
    meanBlueParent ← parent.getBlueIntensityValue()

    cov ← covariance(parent, child1, child2, child3, child4)
    redCovariance ← cov[0]
    greenCovariance ← cov[1]
    blueCovariance ← cov[2]

    redChildVariance ← meanFourValue(specificVariance(child1)[0],
    specificVariance(child2)[0], specificVariance(child3)[0],
    specificVariance(child4)[0])
    greenChildVariance ← meanFourValue(specificVariance(child1)[1],
    specificVariance(child2)[1], specificVariance(child3)[1],
    specificVariance(child4)[1])
    blueChildVariance ← meanFourValue(specificVariance(child1)[2],
    specificVariance(child2)[2], specificVariance(child3)[2],
    specificVariance(child4)[2])

    redParentVariance ← specificVariance(parent)[0]
    greenParentVariance ← specificVariance(parent)[1]
    blueParentVariance ← specificVariance(parent)[2]

    ssimRed ← ((2 * meanRedChild * meanRedParent + c1) * (2 * redCovariance +
    c2)) / ((meanRedChild2 + meanRedParent2 + c1) * (redChildVariance +
    redParentVariance + c2))
    ssimGreen ← ((2 * meanGreenChild * meanGreenParent + c1) * (2 *
    greenCovariance + c2)) / ((meanGreenChild2 + meanGreenParent2 + c1) *
    (greenChildVariance + greenParentVariance + c2))
    ssimBlue ← ((2 * meanBlueChild * meanBlueParent + c1) * (2 *
    blueCovariance + c2)) / ((meanBlueChild2 + meanBlueParent2 + c1) *
    (blueChildVariance + blueParentVariance + c2))

    ssimRGB ← 0.2989 * ssimRed + 0.5787 * ssimGreen + 0.1140 * ssimBlue
```

```

Return ssimRGB
End Procedure

```

Fungsi structuralSimilarityIndex menghitung indeks kesamaan struktural (SSIM) antara sebuah node parent dengan empat node childnya dengan cara menghitung rata-rata intensitas (mean) pada masing-masing channel (merah, hijau, biru) untuk parent dan child, kemudian menghitung kovarians dan variansi pada channel tersebut, dan akhirnya menggabungkan nilai-nilai tersebut menggunakan rumus SSIM yang juga memanfaatkan bobot standar ITU-R BT.601 (0.2989, 0.5787, 0.1140). Konstanta c1 dan c2 masing-masing didefinisikan sebagai  $(0.01 \times 255)^2$  dan  $(0.02 \times 255)^2$ , yang merupakan nilai yang umum digunakan untuk menghindari pembagian dengan nol dan menstabilkan hasil perhitungan pada kondisi kontras rendah. Meskipun demikian, metode SSIM ini sering kali sangat volatil dan tidak merata karena merupakan pengukuran non-linear, sehingga sangat sensitif terhadap perbedaan kecil dalam intensitas dan kontras. Akibatnya, pada rentang angka yang sangat sempit fluktuasi nilai SSIM bisa sangat ekstrim, menghasilkan tingkat kompresi yang sangat tinggi di satu area dan sangat rendah di area lain, meskipun perbedaan yang terukur sangat kecil.

#### 4.6.2 Implementasi Target Kompresi (%)

```

// Bagian dari Main.java

max_iteration ← 30
inputImageSize ← file size of absoluteImageInputAddress
outputImageSize ← -1
delta_change ← -1
if (inputMode ≠ 5) then
    deltaToleranceToTarget ← 100
else
    deltaToleranceToTarget ← 500
expectedOutputSize ← inputImageSize * (1 - compressionPercentageTarget)
surpassTarget ← false
firstIteration ← true

if (inputMode == 1) then
    delta_change ← 8192
else if (inputMode == 2) then
    delta_change ← 63.5
else if (inputMode == 3) then
    delta_change ← 127
else if (inputMode == 4) then
    delta_change ← 4
else if (inputMode == 5) then
    delta_change ← 0.5
else
    delta_change ← -1
if (countCompression is true) then
    treshold ← delta_change / 2

output ← new BufferedImage(width of imageFile, height of imageFile, type of
imageFile)

qt ← new QuadTree(width of imageFile, height of imageFile)

```

```

startTime ← current time in milliseconds

if (countCompression is true) then:
    iteration ← max_iteration
    outputSize ← 0
    countSameOutputSize ← 0

    while ( |outputImageSize - expectedOutputSize| > deltaToleranceToTarget
        AND iteration > 0
        AND countSameOutputSize < 3 ) do:
        qt.compressQuadTree(inputMode, threshold, minimumBlockSize)
        qt.reconstructQuadTree(output)
        outputImageSize ← ProcessImage.getImageSize(output,
        absoluteImageOutputAddress)

        Print "Compression " + (max_iteration - iteration + 1) + "/30 =>
threshold: " + threshold
            + " | compression percentage: " + ((1 - (outputImageSize /
inputImageSize)) * 100) + "%"

        if (outputImageSize > expectedOutputSize) then:
            if (NOT firstIteration AND NOT surpassTarget) then:
                delta_change ← delta_change / (if inputMode == 5 then 1.8
else 2.2)
                threshold ← threshold + delta_change
                surpassTarget ← true
            else if (outputImageSize < expectedOutputSize) then:
                if (NOT firstIteration AND surpassTarget) then:
                    delta_change ← delta_change / (if inputMode == 5 then 1.8
else 2.2)
                    threshold ← threshold - delta_change
                    surpassTarget ← false
                end if

                firstIteration ← false
                iteration ← iteration - 1

                if (outputSize equals outputImageSize) then:
                    countSameOutputSize ← countSameOutputSize + 1
                else:
                    outputSize ← outputImageSize
                    countSameOutputSize ← 0
                end if
            end while
        else:
            qt.compressQuadTree(inputMode, threshold, minimumBlockSize)
            qt.reconstructQuadTree(output)
    end if

```

Pseudocode diatas merupakan bagian dari Main.java, dimana pada bagian tersebut, proses penyesuaian threshold kompresi dilakukan secara iteratif. Proses dimulai dengan inisialisasi variabel-variabel penting (jumlah iterasi maksimum, ukuran file input, toleransi delta, expected output size, dan nilai delta\_change yang ditentukan berdasarkan input mode), kemudian diikuti dengan loop yang akan terus mengompres dan menrekonstruksi citra sehingga perbedaan ukuran antara output dan target berada dalam toleransi atau jumlah iterasi habis. Di tiap iterasi, jika ukuran output melebihi target, threshold akan dinaikkan (dan delta\_change dikurangi berdasarkan mode), sedangkan jika ukuran output kurang dari target, threshold akan diturunkan (delta\_change juga akan dikurangi dengan pembagian tergantung metode perhitungan error). Constraint penerapan disini meliputi batasan iterasi maksimum (30

iterasi) dan batasan perubahan threshold (dengan kondisi stop ketika perubahan ukuran output konstan selama beberapa iterasi) untuk menjaga kestabilan proses kompresi.

#### 4.6.1 Implementasi GIF

**Void reconstructQuadTreeForGIF()**

```

Procedure reconstructQuadTreeForGIF(output, currentDepth, depthLimit)
    if (isLeaf() OR currentDepth ≥ depthLimit) then
        for each y from startY() to endY()-1 do:
            for each x from startX() to endX()-1 do:
                newColor ← new Color(getRedIntensityValue(),
getGreenIntensityValue(), getBlueIntensityValue())
                output.setRGB(x, y, newColor.getRGB())
            end for
        end for
        return
    else
        if (northEast exists) then
            call northEast.reconstructQuadTreeForGIF(output, currentDepth +
1, depthLimit)
        end if
        if (northWest exists) then
            call northWest.reconstructQuadTreeForGIF(output, currentDepth +
1, depthLimit)
        end if
        if (southEast exists) then
            call southEast.reconstructQuadTreeForGIF(output, currentDepth +
1, depthLimit)
        end if
        if (southWest exists) then
            call southWest.reconstructQuadTreeForGIF(output, currentDepth +
1, depthLimit)
        end if
    end if
End Procedure

```

```

// GIF Write & Save (di Main.java)
maxDepth ← qt.getDepth()
frames ← new empty List
for depth from 1 to maxDepth do:
    frameImage ← new BufferedImage(width of imageFile, height of imageFile,
type of imageFile)
    call qt.reconstructQuadTreeForGIF(frameImage, currentDepth = 1,
depthLimit = depth)
    frameCopy ← new BufferedImage(same dimensions and type as frameImage)
    use frameCopy.getGraphics().drawImage(frameImage, 0, 0, null) to copy
    frameImage into frameCopy
    add frameCopy to frames
end for

```

Kedua pseudocode ini menjelaskan secara terpisah dua proses utama dalam pembentukan GIF dari QuadTree. Pseudocode pertama, menggambarkan metode rekursif yang memproses setiap node QuadTree: jika node tersebut merupakan leaf atau kedalaman saat ini mencapai batas, maka bagian citra yang bersesuaian diisi dengan warna yang dihitung dari intensitas rata-rata (RGB), jika tidak, maka prosedur akan memanggil dirinya sendiri pada keempat child dengan penambahan nilai

currentDepth. Pseudocode kedua merupakan potongan code dari Main.java dimana proses pembuatan GIF secara keseluruhan terjadi. Setiap iterasi dalam for loop, dilakukan pengambilan frame yang disimpan dalam sebuah list dengan GifWriter per kedalaman node, jika QuadTree memiliki 11 maximal kedalaman node, maka Gif akan mengandung 11 Frame yang berbeda menurut tingkatan nodenya.

# BAB VI

## TEST CASE

- #Test Case 1 (Variance no percentage target & mid threshold (.png))

input	output
	
Absolute image path : C:\Users\BRIAN RICARDO TANOSH\Downloads\ITB\Semester 4\Algorithm Strategy\Tuci12_13523126\test\mengong.png Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) mode : (1/2/3/4/5) : 1  Threshold Variance (Ideal range [0...16384]) : 108 Minimum Block Size (Integer) : 8 Percentage Compression Target : 8 Absolute Output Image Address : (png, jpg, jpeg) : C:\Users\BRIAN RICARDO TANOSH\Downloads\ITB\Semester 4\Algorithm Strategy\Tuci12_13523126\test\testCase1.png Absolute gif Address : C:\Users\BRIAN RICARDO TANOSH\Downloads\ITB\Semester 4\Algorithm Strategy\Tuci12_13523126\test\testCase1.gif	Save image success! Save gif success!  Execution Time : 535ms Image before compression size : 686982 bytes Image after compression size : 132085 bytes Compression percentage : 80.77% QuadTree depth : 11 QuadTree nodes : 28893
<b>GIF</b>	
	

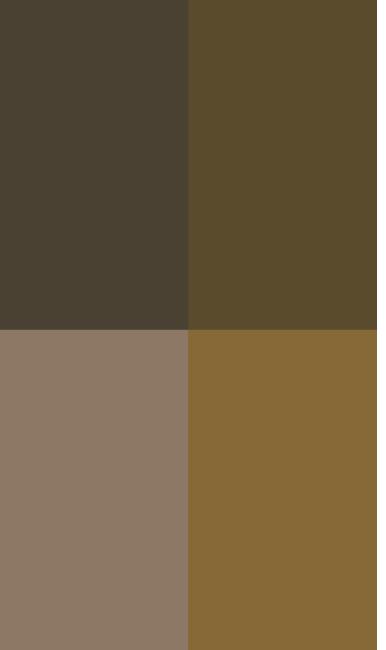
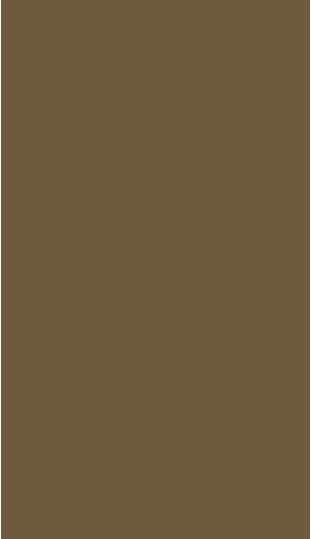
- #Test Case 2 (Mean Absolute Deviation no percentage target & low threshold (.jpg))

input	output
	
<pre>Absolute Image path : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_185232120\test\kodok.jpg Select the following Error Measurement Methods :     1. Absolute Error Measurement Methods :         1. Variance         2. Mean Absolute Deviation (MAD)         3. Max Pixel Difference         4. Entropy         5. Structural Similarity Index (SSIM)         6. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 2  Threshold Mean Absolute Deviation (Ideal range [0..127.5]) : 1 Minimum Output Size (Default) : 50000 Percentage Compression Target : 0 Absolute Output Image Address (png, jpg, jpg) : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_185232120\test\testCase2.jpg Absolute gif Address : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_185232120\test\testCase2.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 1072ms Image before compression size : 59753 bytes Image after compression size : 58582 bytes Compression percentage : 1.96% QuadTree depth : 12 QuadTree nodes : 223669</pre>
<b>GIF</b>	

- #Test Case 3 (Max Pixel Difference no percentage target & low threshold (.jpeg))

input	output
	
Absolute Image path : C:\Users\BRIAN RICARDO TAWI\Downloads\ITB\Semester 4\Algorithm Strategy\Uc112_3523126\testsetan.jpg Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) mode : (1,2,3,4,5) : 3  Threshold Max Pixel Difference (Ideal range [0..255]) : 10 Minimum Block Size (Integer) : 8 Percentage Compression Target : 8 Absolute Output Image Address : (png, jpg, jpeg) : C:\Users\BRIAN RICARDO TAWI\Downloads\ITB\Semester 4\Algorithm Strategy\Uc112_3523126\test\testCase3.jpg Absolute gif Address : c:\users\brian ricardo tawi\downloads\itb\semester 4\algorithm strategy\uc112_3523126\test\testCase3.gif	Save image success! Save gif success!  Execution Time : 590ms Image before compression size : 63499 bytes Image after compression size : 53830 bytes Compression percentage : 15.23% QuadTree depth : 11 QuadTree nodes : 82433
<b>GIF</b>	
	

- #Test Case 4 (Entropy no percentage target & high threshold (.png))

input	output
	
<pre>Absolute Image path : C:\Users\BRIAN RICARDO TAMIN\Downloads\ITB\Semester 4\Algorithm Strategy\Turtle_13523120\test\madflow.png Select the following Error Measurement Methods : 1. L1 Norm 2. Mean Absolute Deviation (MAD) 3. Mean Absolute Difference (MAD) 4. Root Mean Square Difference 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 4  Threshold Entropy (Ideal range [0..8]) : 8 Minimum Block Size (Integer) : 20 Percentage Compression Target : 8 Absolute Image Address (See, Use) : C:\Users\BRIAN RICARDO TAMIN\Downloads\ITB\Semester 4\Algorithm Strategy\Turtle_13523120\test\testCase4.jpg Absolute gif Address : C:\Users\BRIAN RICARDO TAMIN\Downloads\ITB\Semester 4\Algorithm Strategy\Turtle_13523120\test\testCase4.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 190ms Image before compression size : 304769 bytes Image after compression size : 9071 bytes Compression percentage : 97.02% QuadTree depth : 2 QuadTree nodes : 5</pre>
<b>GIF</b>	
	

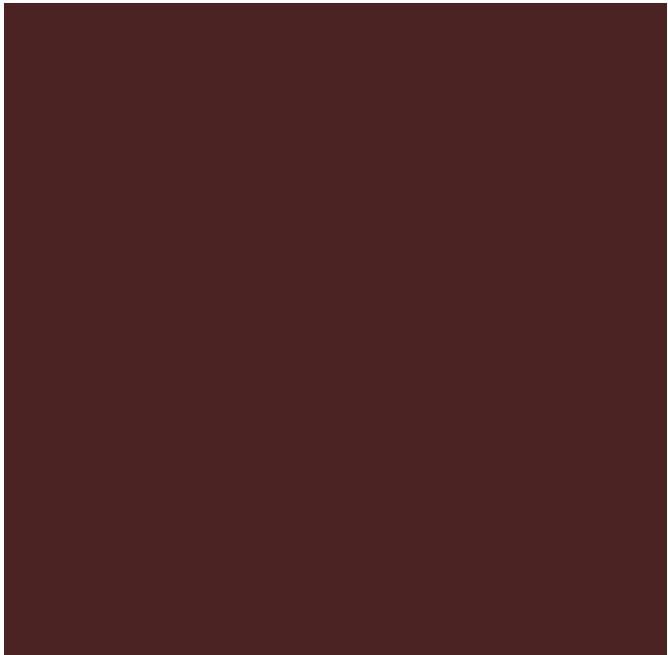
- #Test Case 5 (Structural Similarity Index no percentage target & low threshold (.png))

input	output
	
<pre>Absolute image path : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc12_13523126\test\badNow.png Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Mean Squared Error 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 5  Threshold Structural Similarity Index (ideal range [-1..1]) : .5 Minimum Block Size (Integer) : 15 Percentage Compression Target : 0 Absolute Output Image Address (png, jpg, jpeg) : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc12_13523126\test\testCases.png Absolute gif Address : C:\Users\BRIAN RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc12_13523126\test\testCases.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 4684ms Image before compression size : 304769 bytes Image after compression size : 176412 bytes Compression percentage : 42.12% QuadTree depth : 9 QuadTree nodes : 72709</pre>
<b>GIF</b>	

- #Test Case 6 (Variance with percentage target {34.56%} (.png))

input	output
	
<pre>Absolute Image path : C:\Users\BRUNA RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc11_1352326\testimage.png Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 1  Threshold Variance (Ideal, range [0...10384]) : 1235 Minimum Block Size (Integer) : 5 Percentage Compression Target : 3456 Absolute Output Image Address (jpg, jpg, jpeg) : C:\Users\BRUNA RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc11_1352326\test\testCase1.png Absolute gif Address : C:\Users\BRUNA RICARDO TAVIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc11_1352326\test\testCase1.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 17141ms Image before compression size : 686982 bytes Image after compression size : 449527 bytes Compression percentage : 34.56% QuadTree depth : 10 QuadTree nodes : 182077</pre>
GIF	

- #Test Case 7 (Mean Absolute Deviation with percentage target {12.63%} (.jpeg))

input	output
	
<pre>Absolute Image path : C:\Users\BRIAN RICARDO TANIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc112_13523126\test\Rose.jpg Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 2  Threshold Mean Absolute Deviation (ideal range [0..127.5]) : 45 Minimum Block Size (integer) : 8 Percentage Compression Target: 0.1611 Absolute Output Image Address (.png, .jpg, .jpeg) : C:\Users\BRIAN RICARDO TANIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc112_13523126\test\testCase7.jpg Absolute gif Address : C:\Users\BRIAN RICARDO TANIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tuc112_13523126\test\testCase7.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 12394ms Image before compression size : 237520 bytes Image after compression size : 190396 bytes Compression percentage : 19.84% QuadTree depth : 11 QuadTree nodes : 1398101</pre>
<b>GIF</b>	
	

- #Test Case 8 (Max Pixel Difference with percentage target {100%} (.png) )

input	output
	
<pre>Absolute image path : C:\Users\BRIAN RICARDO TAMIN\Downloads\1TB\Semester 4\Algorithm Strategy\Lucii_13523120\test\Mantra.png Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) node (1/2/3/4/5) : 3  Threshold Max Pixel Difference (Ideal range [0..255]) : 200 Minimum Block Size (Integer) : 5 Percentage Compression Target: 1 Absolute Output Image Address (png, jpg, jpeg) : C:\Users\BRIAN RICARDO TAMIN\Downloads\1TB\Semester 4\Algorithm Strategy\Lucii_13523120\test\testCase8.png Absolute gif Address : C:\Users\BRIAN RICARDO TAMIN\Downloads\1TB\Semester 4\Algorithm Strategy\Lucii_13523120\test\testCase8.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 22ms Image before compression size : 72774 bytes Image after compression size : 625 bytes Compression percentage : 99.14% QuadTree depth : 1 QuadTree nodes : 1</pre>
<b>GIF</b>	
	

- #Test Case 9 (Entropy with percentage target {54.56%} (.jpg))

input	output
	
Absolute Image path : C:\Users\BRIAN RICARDO TANTIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_13523120\test\test.jpg Select the following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Correlation 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5) : 4  Threshold Entropy (Ideal range [0..4]) : 5 Minimum Block Size (Integer) : 5 Percentage Compression Target : 0.5456 Absolute Output Image Address (.png, .jpg, .jpeg) : C:\Users\BRIAN RICARDO TANTIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_13523120\test\testCase9.jpg Absolute gif Address : C:\Users\BRIAN RICARDO TANTIN\Downloads\ITB\Semester 4\Algorithm Strategy\Tucil2_13523120\test\testCase9.gif	Save image success! Save gif success!  Execution Time : 13563ms Image before compression size : 325132 bytes Image after compression size : 147747 bytes Compression percentage : 54.56% QuadTree depth : 10 QuadTree nodes : 68981
<b>GIF</b>	
	

- #Test Case 10 (Structural Similarity Index with percentage target {63.54%} (.png))

input	output
	
<pre>Absolute Image path : C:\Users\BRIAN RICARDO TAMON\Downloads\ITB\Semester 4\Algoritma Strategi\Tuc112_13523126\test\WS.png Select one following Error Measurement Methods : 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy 5. Structural Similarity Index (SSIM) mode (1/2/3/4/5): 5  Threshold Structural Similarity Index (ssim) range [-1..1] : 1 Minimum Block Size (Integer): 6 Percentage Compression Target: 0.6354 Absolute Output Image Address (.png, .jpg, .Steg): C:\Users\BRIAN RICARDO TAMON\Downloads\ITB\Semester 4\Algoritma Strategi\Tuc112_13523126\test\testCase10.png Absolute gif Address : C:\Users\BRIAN RICARDO TAMON\Downloads\ITB\Semester 4\Algoritma Strategi\Tuc112_13523126\test\testCase10.gif</pre>	<pre>Save image success! Save gif success!  Execution Time : 37898ms Image before compression size : 454296 bytes Image after compression size : 173996 bytes Compression percentage : 61.70% QuadTree depth : 9 QuadTree nodes : 76869</pre>
<b>GIF</b>	
	

## **BAB VII**

### **KESIMPULAN & SARAN**

#### **7.1 Kesimpulan**

Algoritma yang digunakan untuk melakukan compression pada image adalah *divide and conquer* menggunakan struktur data hirarkis quadtree. Metode ini cukup sederhana dan mudah untuk diimplementasikan dalam kompresi gambar melalui 4 node child dengan patokan threshold dan minimum block size.

#### **7.2 Saran**

Tugas kecil ini mewajibkan penggunaan algoritma *divide and conquer* dengan struktur data quadtree sehingga tidak memerlukan terlalu banyak memory dan computing power yang cukup signifikan dibanding tugas kecil sebelumnya. Menurut saya tugas kecil ini sudah sangat jelas, informatif, dan menambah pemahaman saya mengenai struktur data baru yaitu quadtree.

## **BAB VIII**

### **DAFTAR PUSTAKA**

- [1] Wang, Z., & Bovik, A. C. (2002). "A Universal Image Quality Index". *IEEE Transactions on Signal Processing*, 48(7), 1–11 (<https://ieeexplore.ieee.org/document/993146>) [Accessed: Apr. 2, 2025].
- [2] Wang, Z., & Bovik, A. C. (2004). "Multiscale Structural Similarity for Image Quality Assessment". *IEEE Transactions on Image Processing*, 13(4), 600–612 (<https://ieeexplore.ieee.org/document/1263918>) [Accessed: Apr. 2, 2025].

# LAMPIRAN

Github Repository : [https://github.com/brii26/Tucil2\\_13523126](https://github.com/brii26/Tucil2_13523126)

Google Docs link : <https://bit.ly/laporanStimaTucil2> (untuk melihat .gif pada laporan)

Tabel Poin

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. <b>[Bonus]</b> Implementasi persentase kompresi sebagai parameter tambahan	✓	
6. <b>[Bonus]</b> Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	✓	
7. <b>[Bonus]</b> Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar	✓	
8. Program dan laporan dibuat sendiri	✓	