

Laporan Tugas Kecil 2
IF2211 Strategi Algoritma
Kompresi Gambar Dengan Metode Quadtree
Semester II Tahun 2024/2025



Disusun oleh :
Clarissa Nethania Tambunan (13523016)
Shannon Aurellius Anastasya Lie (13523019)

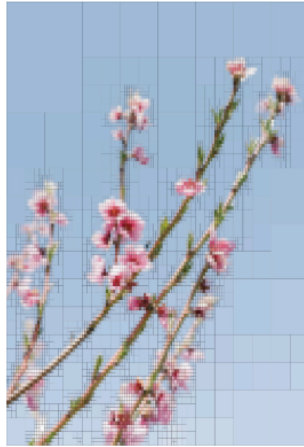
Dosen Pengampu:
Dr. Nur Ulfa Maulidevi, S.T, M.Sc.
Dr. Ir. Rinaldi Munir, M.T.
Menterico Adrian, S.T, M.T.

Program Studi Teknik Informatika
Sekolah Teknik Elektro Dan Informatika
Institut Teknologi Bandung
2025

DAFTAR ISI

DAFTAR ISI.....	2
BAB I DESKRIPSI TUGAS.....	3
BAB II LANDASAN TEORI.....	8
2.1. Algoritma Divide and Conquer.....	8
2.2. Alur Kerja Program.....	8
BAB III ANALISIS ALGORITMA DAN IMPLEMENTASI.....	9
3.1. Implementasi Algoritma Divide and Conquer.....	9
3.1.1. File Input.java.....	9
3.1.2. File Output.java.....	10
3.1.3. File GIFGenerator.java.....	10
3.1.4. File ImageCompressor.java.....	11
3.1.5. File Main.java.....	12
3.2. Notasi Pseudocode.....	13
3.2.1. File GIFGenerator.java.....	13
3.2.2. File ImageCompressor.java.....	14
3.2.3. File Input.java.....	18
3.2.4. File Output.java.....	20
3.2.5. File Main.java.....	20
3.3. Analisis Kompleksitas Algoritma.....	22
BAB IV PENGUJIAN.....	23
4.1. Test Case 1.....	23
4.2. Test Case 2.....	25
4.3. Test Case 3.....	27
4.4. Test Case 4.....	30
4.5. Test Case 5.....	32
4.6. Test Case 6.....	34
4.7. Test Case 7.....	37
4.8. Test Case 8.....	39
4.9. Test Case 9.....	41
4.10. Test Case 10.....	44
BAB V KESIMPULAN DAN SARAN.....	48
5.1. Kesimpulan.....	48
5.2. Saran.....	48
DAFTAR PUSTAKA.....	49
LAMPIRAN.....	49

BAB I DESKRIPSI TUGAS



Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

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

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



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar

(Sumber: https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsmbgNBFrYkxyG5dA.gif)

****Cek sumber untuk melihat animasi GIF****

Ilustrasi kasus :

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (*Divide and Conquer*):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- a) **Metode perhitungan variansi:** pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada *Tabel 1*.
- b) **Threshold variansi:** nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
- c) **Minimum block size:** ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan *Error*

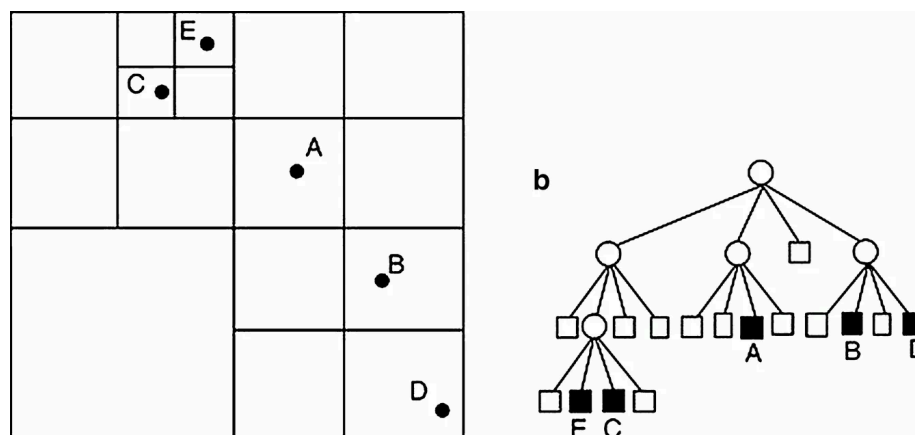
Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai *Tabel 1*.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- ❖ Jika variansi **di atas threshold** (*cek kasus khusus untuk metode bonus*), ukuran blok lebih besar dari **minimum block size**, dan ukuran blok setelah dibagi menjadi empat **tidak kurang dari minimum block size**, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.

- ❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.
4. Normalisasi Warna
Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.
 5. Rekursi dan Penghentian
Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:
 - ❖ *Error* blok berada di bawah threshold.
 - ❖ Ukuran blok setelah dibagi menjadi empat kurang dari *minimum block size*.
 6. Penyimpanan dan Output
Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyork/quadtrees-for-image-processing-302536c95c00>)

Parameter:

1. *Error Measurement Methods* (Metode Pengukuran *Error*)

Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika *error* dalam blok melebihi ambas batas (*threshold*), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Tabel 1. Metode Pengukuran *Error*

Metode	Formula
<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok

	$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c
Entropy	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok $P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)
[Bonus] Structural Similarity Index (SSIM) (Referensi tambahan)	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

2. Threshold (Ambang Batas)

Threshold adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.

3. Minimum Block Size (Ukuran Blok Minimum)

Minimum block size (luas piksel) adalah ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Jika ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah ukuran minimum yang telah dikonfigurasi, maka blok tersebut tidak akan dibagi lebih lanjut, meskipun *error* masih di atas threshold.

4. Compression Percentage (Persentase Kompresi) [BONUS]

Persentase kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

$$\text{Persentase Kompresi} = \left(1 - \frac{\text{Ukuran Gambar Terkompresi}}{\text{Ukuran Gambar Asli}}\right) \times 100\%$$

BAB II

LANDASAN TEORI

2.1. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* terdiri dari dua kata yaitu “*divide*” dan “*conquer*”. Kata “*divide*” berarti membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama). Sedangkan, “*conquer*” memiliki makna menyelesaikan (solve) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar). Algoritma *divide and conquer* akan melakukan *combine* setelah proses “*divide*” dan “*conquer*” selesai. Kata “*combine*” bermakna menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Objek persoalan yang dibagi berupa masukan (*input*) atau *instances* persoalan yang berukuran n seperti tabel (larik), matriks, eksponen, polinom, dll, bergantung persoalannya. Tiap-tiap upa-persoalan memiliki karakteristik yang sama (*the same type*) dengan karakteristik persoalan semula namun berukuran lebih kecil sehingga metode *Divide and Conquer* lebih natural diungkapkan dalam skema rekursif.

2.2. Alur Kerja Program

Berikut merupakan alur kerja program.

1. [INPUT] alamat absolut gambar yang akan dikompresi.
2. [INPUT] metode perhitungan *error* (gunakan penomoran sebagai input).
3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
4. [INPUT] ukuran blok minimum.
5. [INPUT] Target persentase kompresi (floating number, $1.0 = 100\%$), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
6. [INPUT] alamat absolut gambar hasil kompresi.
7. [INPUT] alamat absolut gif (bonus).
8. [OUTPUT] waktu eksekusi.
9. [OUTPUT] ukuran gambar sebelum.
10. [OUTPUT] ukuran gambar setelah.
11. [OUTPUT] persentase kompresi.
12. [OUTPUT] kedalaman pohon.
13. [OUTPUT] banyak simpul pada pohon.
14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

BAB III

ANALISIS ALGORITMA DAN IMPLEMENTASI

3.1. Implementasi Algoritma *Divide and Conquer*

Penerapan algoritma *Divide and Conquer* dapat digunakan dalam konteks kompresi gambar dengan *Quadtree*. Penggunaan utamanya dapat dibagi menjadi tiga yaitu:

1. *Divide* (Pembagian Blok)

Gambar yang diterima melalui *input* dibagi menjadi blok persegi. Jika blok tersebut tidak seragam (misalnya, nilai *error*-nya melebihi *threshold*), maka blok tersebut dibagi lagi menjadi empat bagian (kuadran): kiri-atas, kanan-atas, kiri-bawah, dan kanan-bawah. Hal ini dilakukan melalui fungsi kompresi yang memanggil dirinya sendiri secara rekursif.

2. *Conquer* (Kompresi Sub-blok)

Masing-masing dari keempat sub-blok tersebut kemudian diperiksa kembali apakah mereka masih perlu dibagi. Jika sudah cukup seragam (berdasarkan metode *error* dan *threshold* yang dipilih), maka blok tersebut tidak dibagi lagi (berhenti melakukan pembagian), dan diwakili oleh satu warna rata-rata.

3. *Combine* (Penggabungan Hasil)

Setelah semua blok telah diproses, gambar hasil kompresi dibentuk dari representasi tiap simpul daun pada pohon *Quadtree*. Proses ini juga menghasilkan gambar kompresi akhir dan animasi GIF yang menunjukkan transisi kompresi dari akar hingga kedalaman terakhir.

Berikut merupakan implementasi kompresi gambar menggunakan algoritma *Divide and Conquer* dalam file-file menggunakan bahasa pemrograman Java.

3.1.1. File Input.java

- **Atribut**

Atribut	Deskripsi
public String inputPath	<i>Path</i> menuju file gambar asli.
public String outputPath	<i>Path</i> untuk menyimpan gambar hasil kompresi.
public String gifPath	<i>Path</i> untuk menyimpan file animasi GIF.
public int threshold	Batas toleransi error untuk membagi blok.
public int minBlockSize	Ukuran minimum blok yang boleh dibagi lagi.
public String errorMethod	Metode yang digunakan untuk menghitung error.

- **Konstruktor**

Tidak terdapat konstruktor.

- **Metode**

Metode	Deskripsi
public void inputAll(String[] args)	Membaca seluruh argumen dari command-line dan menginisialisasi atribut.

3.1.2.File Output.java

- **Atribut**

Tidak terdapat atribut.

- **Konstruktor**

Tidak terdapat konstruktor.

- **Metode**

Metode	Deskripsi
public static void printStats(long startTime, long endTime, Input input, int depth, int nodeCount)	Menampilkan statistik waktu eksekusi, ukuran file sebelum dan sesudah kompresi, persentase kompresi, serta info struktur pohon.

3.1.3.File GIFGenerator.java

- **Atribut**

Atribut	Deskripsi
protected ImageWriter gifWriter	<i>Writer</i> internal untuk menghasilkan file GIF.
protected ImageWriteParam imageWriteParam	Parameter penulisan gambar.
protected IIOMetadata imageMetaData	Metadata yang mengatur <i>frame</i> GIF (<i>delay</i> , <i>loop</i> , dll).

- **Konstruktor**

Konstruktor	Deskripsi
public GifSequenceWriter(ImageOutputStream outputStream, int imageType, int delay, boolean loop)	Inisialisasi <i>writer</i> dan metadata GIF dengan <i>delay</i> dan opsi <i>loop</i> .

- **Metode**

Metode	Deskripsi
public void writeToSequence(RenderedImage img)	Menulis sebuah <i>frame</i> ke dalam sequence GIF.
public void close()	Menutup <i>writer</i> dan mengakhiri <i>sequence</i> GIF.
private static IIOMetadataNode getNode(IIOMetadataNode rootNode, String nodeName)	Mengambil atau membuat node metadata berdasarkan nama node.

3.1.4.File ImageCompressor.java

- **Atribut**

Atribut	Deskripsi
private BufferedImage image	Gambar input yang akan dikompresi.
private int width	Lebar gambar.
private int height	Tinggi gambar.
private String errorMethod	Menyimpan pilihan metode perhitungan <i>error</i> .
private int threshold	Ambang batas <i>error</i> untuk memutuskan pembagian blok.
private int minBlockSize	Ukuran minimum blok (basis kasus rekursi).
private List<BufferedImage> gifFrames	List <i>frame-frame</i> per kedalaman untuk animasi GIF.
private int maxDepth	Kedalaman maksimal yang dicapai pohon <i>quadtree</i> .
private int nodeCount	Total simpul yang terbentuk pada proses kompresi.

- **Konstruktor**

Konstruktor	Deskripsi
public ImageCompressor(BufferedImage	Inisialisasi objek dengan gambar dan

image, String errorMethod, int threshold, int minBlockSize)	parameter kompresi.
---	---------------------

- **Metode**

Metode	Deskripsi
public BufferedImage compress()	Melakukan kompresi menggunakan algoritma <i>Quadtree</i> dan mengembalikan gambar hasil kompresi.
private BufferedImage compressBlock(int x, int y, int size, int depth)	Fungsi rekursif <i>divide and conquer</i> yang membagi blok jika <i>error</i> melebihi threshold.
private boolean shouldSplit(int x, int y, int size)	Menentukan apakah sebuah blok perlu dibagi lebih lanjut.
private int[] getAverageColor(int x, int y, int size)	Menghitung rata-rata warna blok.
private double calculateError(int x, int y, int size, int[] avgColor)	Menghitung nilai <i>error</i> blok terhadap rata-rata warnanya.
private BufferedImage drawBlock(int x, int y, int size, int[] color, BufferedImage canvas)	Menggambar blok berwarna rata-rata pada kanvas hasil kompresi.
public List<BufferedImage> getGifFrames()	Mengembalikan <i>frame-frame</i> yang disimpan selama proses kompresi.
public int getMaxDepth()	Mendapatkan kedalaman maksimum pohon hasil kompresi.
public int getNodeCount()	Mendapatkan jumlah total simpul pohon hasil kompresi.

3.1.5.File Main.java

- **Atribut**
Tidak terdapat atribut.
- **Konstruktor**
Tidak terdapat konstruktor.
- **Metode**

Metode	Deskripsi
--------	-----------

<pre>public static void main(String[] args)</pre>	<p>Fungsi utama program: membaca input, memulai proses kompresi dengan algoritma <i>quadtree</i>, menyimpan hasil (gambar dan GIF), serta menampilkan statistik.</p>
---	--

3.2. Notasi Pseudocode

3.2.1. File GIFGenerator.java

```

Class GIFGenerator:
    Procedure saveGif(frames:List of Image, gifPath:String):
        try (open output stream to gifPath
            create GifSequenceWriter with output stream, image type RGB, delay
500ms, looping true
            for each frame in frames:
                writer.writeToSequence(frame)
            writer.close()
            output.close()

class GifSequenceWriter:
    field gifWriter : ImageWriter
    field imageWriteParam : ImageWriteParam
    field imageMetaData : IIOMetadata

    constructor GifSequenceWriter(outputStream:ImageOutputStream, imageType:integer,
delay:integer, loop:boolean):
        gifWriter <- ImageIO.getImageWritersBySuffix("gif").next()
        imageWriteParam <- gifWriter.getDefaultWriteParam()
        ImageTypeSpecifier imageTypeSpecifier <-
ImageTypeSpecifier.createFromBufferedImageType(imageType)
        imageMetaData <- gifWriter.getDefaultImageMetadata(imageTypeSpecifier,
imageWriteParam)
        String metaFormatName <- imageMetaData.getNativeMetadataFormatName()
        IIOMetadataNode root <- (IIOMetadataNode)
imageMetaData.getAsTree(metaFormatName)
        IIOMetadataNode gce <- getNode(root, "GraphicControlExtension")
        gce.setAttribute("disposalMethod", "none")
        gce.setAttribute("userInputFlag", "FALSE")
        gce.setAttribute("transparentColorFlag", "FALSE")
        gce.setAttribute("delayTime", Integer.toString(delay / 10))
        gce.setAttribute("transparentColorIndex", "0")
        IIOMetadataNode appExtensionsNode <- getNode(root,
"ApplicationExtensions");
        IIOMetadataNode appExtensionNode <- new
IIOMetadataNode("ApplicationExtension")
        appExtensionNode.setAttribute("applicationID", "NETSCAPE")
        appExtensionNode.setAttribute("authenticationCode", "2.0")
        byte[] loopBytes = new byte[]{0x1, (byte) (loop ? 0 : 1), 0}
        appExtensionNode.setUserObject(loopBytes)
        appExtensionsNode.appendChild(appExtensionNode)
        root.appendChild(appExtensionsNode)
        imageMetaData.setFromTree(metaFormatName, root)
        gifWriter.setOutput(outputStream)
        gifWriter.prepareWriteSequence(null)

    Procedure writeToSequence(image):
        gifWriter.writeToSequence(new javax.imageio.IIOImage(img, null,
imageMetaData), imageWriteParam)

```

```
Procedure close() throws IO Exception:
    gifWriter.endWriteSequence()
```

```
Function getNode(rootNode:IIOMetadataNode, nodeName:string)-> IIOMetadataNode
    i traversal [0..rootNode.getLength()]
        if (rootNode.item(i).getNodeName().equalsIgnoreCase(nodeName)) then
            -> (IIOMetadataNode) rootNode.item(i)
    IIOMetadataNode node = new IIOMetadataNode(nodeName)
    rootNode.appendChild(node)
    -> node
```

3.2.2.File ImageCompressor.java

```
Class ImageCompressor:
    Class QuadTreeNode:
        x, y, width, height : integer
        color : Color
        isLeaf : boolean
        QuadTreeNode[] children

        COnstructor QuadTreeNode(x:integer, y:integer, width:integer,
height:integer):
            this.x <- x
            this.y <- y
            this.width <- width
            this.height <- height
            this.isLeaf <- false
            this.children <- new QuadTreeNode[4]

        Integer nodeCount <- 0
        Integer maxDepth <- 0
        gifframes <- list kosong untuk menyimpan frame GIF
        depthLevels <- list of list untuk menyimpan node di setiap level

        Function compress(image:BufferedImage, errorMethod:integer, threshold:float,
minBlockSize:integer) -> QuadTreeNode
            nodeCount <- 0
            maxDepth <- 0
            gifFrames.clear()
            depthLevels.clear()

            QuadTreeNode root <- new QuadTreeNode(0, 0, image.getWidth(),
image.getHeight())
            buildQuadTree(image, 0, 0, image.getWidth(), image.getHeight(), threshold,
errorMethod, minBlockSize, 0, root, root)
            i traversal [depth<-0 .. depth<depthLevels.size()]
                BufferedImage frame <- new BufferedImage(image.getWidth(),
image.getHeight(), BUfferedImage.TYPE_INT_RGB)
                Graphics2D g <- frame.createGraphics()
                renderByDepth(g, root, depth)
                g.dispose()
                gifframes.add(frame)
            gifframes.add(image)
            -> root

        Function render(root:QuadTreeNode, width:integer, height:integer) ->
BufferedImage
            BufferedImage img <- new BufferedImage(width, height,
BufferedImage.TYPE_INT_RGB)
            Graphics2D g <- img.createGraphics()
```

```

renderNode(g, root)
g.dispose()
-> img

Function getGifFrames() -> List of BufferedImage
-> gifFrames

Function getNodeCount() -> integer
-> nodeCount

Function getMaxDepth() -> integer
-> maxDepth

Function buildQuadTree(img:BufferedImage, x:integer, y:integer, width:integer,
height:integer, threshold:float, errorMethod:integer, minSize:integer,
depth:integer, node:QuadTreeNode, root:QuadTreeNode) -> QuadTreeNode
    nodeCount <- nodeCount + 1
    maxDepth <- max(maxDepth, depth)
    while (depthLevels.size() <= depth) do:
        tambah list kosong ke depthLevels
    depthLevels[depth].add(node)

    if (width ≤ minSize and height ≤ minSize) or (getError(img, x, y, width,
height, errorMethod) <= threshold) then
        node.isLeaf <- true
        node.color <- getAverageColor(img, x, y, width, height)
        -> node

    midW <- width / 2
    midH <- height / 2

    node.children[0] <- buildQuadTree(img, x, y, midW, midH, threshold,
errorMethod, minSize, depth + 1, new QuadTreeNode(x, y, midW, midH), root)
    node.children[1] <- buildQuadTree(img, x + midW, y, width - midW, midH,
threshold, errorMethod, minSize, depth + 1, new QuadTreeNode(x + midW, y, width -
midW, midH), root)
    node.children[2] <- buildQuadTree(img, x, y + midH, midW, height - midH,
threshold, errorMethod, minSize, depth + 1, new QuadTreeNode(x, y + midH, midW,
height - midH), root)
    node.children[3] <- buildQuadTree(img, x + midW, y + midH, width - midW,
height - midH, threshold, errorMethod, minSize, depth + 1, new QuadTreeNode(x +
midW, y + midH, width - midW, height - midH), root)
    -> node

Procedure renderByDepth(g:Graphics2D, node:QuadTreeNode,
maxRenderDepth:integer)
    call renderByDepthHelper(g, node, 0, maxRenderDepth)

Procedure renderByDepthHelper(g:Graphics2D, node:QuadTreeNode,
currentDepth:integer, maxDepth:integer):
    if (node = null) then
        return;

    if (currentDepth = maxDepth or node.isLeaf) then
        g.setColor(node.color jika tidak null, else Color.BLACK)
        g.fillRect(node.x, node.y, node.width, node.height)
    else
        for setiap child dalam node.children:
            call renderByDepthHelper(g, child, currentDepth + 1, maxDepth)

Procedure renderNode(g:Graphics2D, node:QuadTreeNode):
    if (node.isLeaf) then

```

```

        g.setColor(node.color)
        g.fillRect(node.x, node.y, node.width, node.height)
    else
        for setiap child dalam node.children:
            if (child ≠ null) then
                call renderNode(g, child)

Function getAverageColor(image:BufferedImage, x:integer, y:integer,
width:integer, height:integer)-> Color
    long r <- 0, g <- 0, b <- 0
    count <- 0
    i traversal [y..y + height and (limited to image height)]
        j traversal [x..x + width and (limited to image width)]
            color c <- new Color(img.getRGB(j,i))
            r = r + c.getRed()
            g = g + c.getGreen()
            b = b + c.getBlue()
            count = count + 1
    -> new Color((int) (r/count), (int) (g / count), (int) (b / count))

Function getError(image:BufferedImage, x:integer, y:integer, width:integer,
height:integer, method:integer)-> Double
    integer pixelWidth <- min(width, image width - x)
    integer pixelHeight <- min(height, image height - y)
    integer pixelCount <- pixelWidth * pixelHeight

    double sumR, sumG, sumB, sumSqR, sumSqG, sumSqB to 0
    double diffR, diffG, diffB as empty lists

    i traversal [y..y + pixelHeight]
        j traversal [x..x + pixelWidth]
            color c <- new Color(img.getRGB(j,i))
            integer r<-c.getRed(), g<-c.getGreen(), b<-c.getBlue()
            sumR = sumR + r, sumG = sumG + g, sumB = sumB + b
            sumSqR = sumSqR + (r*r), sumSqG = sumSqG + (g*g), sumSqB =
sumSqB + (b*b)
            Add r, g, b to diffR, diffG, diffB respectively
    double meanR = sumR / pixelCount
    double meanG = sumG / pixelCount
    double meanB = sumB / pixelCount

    Switch (method):
        Case 1: // Variance
            double varR = (sumSqR / pixelCount) - (meanR * meanR)
            double varG = (sumSqG / pixelCount) - (meanG * meanG)
            Double varB = (sumSqB / pixelCount) - (meanB * meanB)
            -> (varR + varG + varB)/3.0

        Case 2: // Mean Absolute Deviation
            mad = 0
            i traversal [0..pixelCount]
                mad = mad |diffR[i] - meanR| + |diffG[i] - meanG| +
|diffB[i] - meanB|
            -> mad / (3 * pixelCount)

        Case 3: // Maximum Difference
            integer maxDiff <- 0
            i traversal [y..y + pixelHeight]
                j traversal [x..x + pixelWidth]
                    color c <- new Color(img.getRGB(j,i))
                    dr = |c.getRed() - meanR|
                    dg = |c.getGreen() - meanG|

```



```

        db = |c.getBlue() - meanB|
        maxDiff = max(maxDiff, max(dr, max(dg, db)))
    -> maxDiff

Case 4: // Entropy
Initialize freqR, freqG, freqB arrays of size 256 with 0
i traversal [y..y + pixelHeight]
    j traversal [x..x + pixelWidth]
        color c <- new Color(img.getRGB(j,i))
        Increment freqR[c.getRed()], freqG[c.getGreen()],
freqB[c.getBlue()]
    double entropyR, entropyG, entropyB <- 0
    For i from 0 to 255:
        double pR = freqR[i] / pixelCount
        double pG = freqG[i] / pixelCount
        double pB = freqB[i] / pixelCount
        If pR > 0: entropyR = entropyR - pR * log2(pR)
        If pG > 0: entropyG = entropyG - pG * log2(pG)
        If pB > 0: entropyB = entropyB - pB * log2(pB)
    -> (entropyR + entropyG + entropyB)/3.0

Case 5: // Structural Similarity Index Measure (SSIM)
double K1 <- 0.01, K2 <- 0.03, L <- 255
double C1 <- (K1 * L)^2, C2 <- (K2 * L)^2

muX <- 0, muY <- 0
orig <- empty list
i traversal [y..y + pixelHeight]
    j traversal [x..x + pixelWidth]
        color c <- new Color(img.getRGB(j,i))
        double gray <- c.getRed() * 0.299 + c.getGreen() *
0.587 + c.getBlue() * 0.114
        Add gray to origGrays
        muX = muX + gray
muX /= pixelCount
Color avg <- getAverageColor(image, x, y, pixelWidth,
pixelHeight)
double avgGray = avg.getRed() * 0.299 + avg.getGreen() * 0.587 +
avg.getBlue() * 0.114
muY <- avgGray
double sigmaX2, sigmaY2, sigmaXY <- 0
For each xValue in orig:
    sigmaX2 = sigmaX2 + (xValue - muX)^2
    sigmaY2 = sigmaY2 + (avgGray - muY)^2 // Always 0
    sigmaXY = sigmaXY + (xValue - muX) * (avgGray - muY)
sigmaX2 /= pixelCount
sigmaY2 /= pixelCount
sigmaXY /= pixelCount
double ssim <- ((2 * muX * muY + C1) * (2 * sigmaXY + C2))
/((muX^2 + muY^2 + C1) * (sigmaX2 + sigmaY2 + C2))
-> 1 - SSIM

Default:
-> 0

Function ImagesEqual(image1, image2) -> boolean
If (img1.getWidth() ≠ img2.getWidth() or img1.getHeight()
≠ img2.getHeight()) then return False

y traversal [0..img1.getHeight()]
x traversal [0..img1.getWidth()]
If (img1.getRGB(x,y) ≠ img2.getRGB(x,y)) then -> False

```

-> True

3.2.3.File Input.java

```
Class Input:
    inputPath : string
    errorMethod : integer
    threshold : float
    minBlockSize : integer
    targetCompression : float
    outputPath : string
    gifPath : string
    Scanner <- new Scanner from system input
    Procedure inputAll:
        Boolean valid
        do
            valid <- true
            create empty list errors
            Output("Masukkan path gambar (absolut) input: ")
            inputPath <- read line from user
            output("")
            output("Pilihan Metode Error:")
            output("1. Variance")
            output("2. Mean Absolute Deviation (MAD)")
            output("3. Max Pixel Difference")
            output("4. Entropy")
            output("5. Structural Similarity Index Measure (SSIM)")
            output("Masukkan pilihan (1/2/3/4/5): ")
            try
                errorMethod <- parse integer from user input
            Catch (NumberFormatException e)
                errorMethod <- errorMethod - 1
            output("")
            output("Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0
dan 1.")
            output("Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan
255.")

            output("Masukkan threshold: ")
            try
                threshold <- parse float from user input
            Catch (NumberFormatException e)
                threshold <- threshold - 1
            output("")
            output("Masukan ukuran blok minimum: ")
            try
                minBlockSize <- parse integer from user input
            Catch (NumberFormatException e)
                minBlockSize <- minBlockSize - 1
            output("")
            output("Masukan target kompresi (0-1): ")
            try
                targetCompression <- parse float from user input
            Catch (NumberFormatException e)
                targetCompression <- targetCompression - 1
            output("")

            {pathnya tanpa tanda petik dua}
            output("Masukan path gambar (absolut) output: ")
            outputPath <- read line from user
            output("Masukan path GIF (absolut) output: ")

```

```

gifPath <- read line from user

output("")
if (inputPath.isEmpty()) then
  valid <- false
  Add "Path input tidak boleh kosong." to errors
else if (!inputPath.matches("(?i)^.+\\. (jpg|jpeg|png)$")) then
  valid <- false
  Add "Format file input harus .jpg, .jpeg, atau .png." to errors
else
  File inputFile <- new File(inputPath)
  if (!inputFile.exists() or !inputFile.isFile()) then
    valid <- false
    Add "File input tidak ditemukan atau bukan file." to errors

  if (errorMethod < 1 or errorMethod > 5) then
    valid <- false
    Add "Metode error harus antara 1 dan 5." to errors

  if ((errorMethod = 1 or errorMethod = 4 or errorMethod = 5) and
(threshold < 0 or threshold > 1)) then
    valid <- false
    Add "Threshold untuk metode tersebut harus antara 0 dan 1." to
errors

  if ((errorMethod = 2 or errorMethod = 3) and (threshold < 0 or
threshold > 255)) then
    valid <- false
    Add "Threshold untuk metode tersebut harus antara 0 dan 255." to
errors

  if (minBlockSize < 1) then
    valid <- false
    Add "Ukuran blok minimum harus >= 1." to errors

  if (targetCompression < 0 or targetCompression > 1) then
    valid <- false
    Add "Target kompresi harus antara 0 dan 1." to errors

  if (outputPath.isEmpty()) then
    valid <- false
    Add "Path output gambar tidak boleh kosong." to errors
  else if (!outputPath.matches("(?i)^.+\\. (jpg|jpeg|png)$")) then
    valid <- false
    Add "Format file output harus .jpg, .jpeg, atau .png." to errors
  else
    File outputFile <- new File(outputPath)
    File parent <- outputFile.getParentFile()
    if (parent != null and !parent.exists()) then
      valid <- false
      Add "Folder tujuan untuk output gambar tidak ditemukan." to
errors

  if (gifPath.isEmpty()) then
    valid <- false
    Add "Path output GIF tidak boleh kosong." to errors
  else if (!gifPath.matches("(?i)^.+\\. (jpg|jpeg|png)$")) then
    valid <- false
    Add "Format file output GIF harus .gif." to errors
  else
    File gifFile <- new File(gifPath)
    File parent <- gifFile.getParentFile()

```


3.3. Analisis Kompleksitas Algoritma

Algoritma kompresi gambar berbasis QuadTree yang digunakan dalam program ini merupakan implementasi dari algoritma *Divide and Conquer* yaitu gambar direpresentasikan dalam bentuk matriks piksel dua dimensi yang dibagi secara rekursif ke dalam empat bagian kuadran (kiri-atas, kanan-atas, kiri-bawah, kanan-bawah) hingga memenuhi kriteria keseragaman tertentu. Proses ini dilakukan hingga setiap blok gambar homogen atau mencapai ukuran blok minimum yang telah ditentukan. Berikut analisis kompleksitas algoritma dari program yang telah diimplementasikan.

1. Kompleksitas Waktu

Kompleksitas waktu dari algoritma ini dalam kasus terburuk adalah $O(n^2 \log n)$ di mana $n \times n$ adalah dimensi gambar yang terjadi ketika gambar memiliki banyak detail atau variasi warna yang tinggi sehingga tidak dapat dikompresi dan setiap piksel pada akhirnya menjadi node tersendiri dalam pohon QuadTree, tetapi dalam kasus bagus kompleksitasnya dapat menurun menjadi $o(\log n)$ tergantung seberapa cepat pembagian berhenti.

2. Kompleksitas Ruang

Kompleksitas ruang algoritma ini bergantung pada jumlah simpul (node) dalam pohon Quadtree. Dalam kasus terburuk, ketika semua blok perlu dipecah hingga unit terkecil, jumlah simpul mendekati 4^d di mana d adalah kedalaman maksimum pohon, yang dalam kasus ekstrim bisa mencapai $O(n^2)$ node. Namun, dalam kasus ideal dengan banyak area seragam, struktur pohon jauh lebih kecil dan efisien, menjadikannya hemat ruang.

3. Efektivitas Kompresi

Eksperimen menunjukkan bahwa algoritma ini sangat efektif dalam mengkompresi gambar yang memiliki banyak area dengan warna seragam atau sedikit variasi. Gambar-gambar seperti ilustrasi kartun atau gambar grayscale sederhana cenderung menghasilkan rasio kompresi yang tinggi dengan kedalaman pohon yang rendah. Sebaliknya, pada gambar dengan noise tinggi atau gradien warna kompleks, efektivitas kompresi menurun karena pohon menjadi lebih dalam dan node menjadi lebih banyak.

4. Waktu Eksekusi

Berdasarkan pengujian, waktu eksekusi meningkat seiring dengan meningkatnya resolusi gambar dan menurunnya threshold karena semakin banyak memecahkan blok gambar yang harus dilakukan.

BAB IV PENGUJIAN

4.1. Test Case 1

test1.jpg



Tampilan pertama saat program baru dijalankan

Computer Graphics

Haiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Pilihan Metode Error:

- Masukkan pilihan (1/2/3/4/5):

Masukkan threshold: 0.5

Masukkan ukuran blok minimum: 100

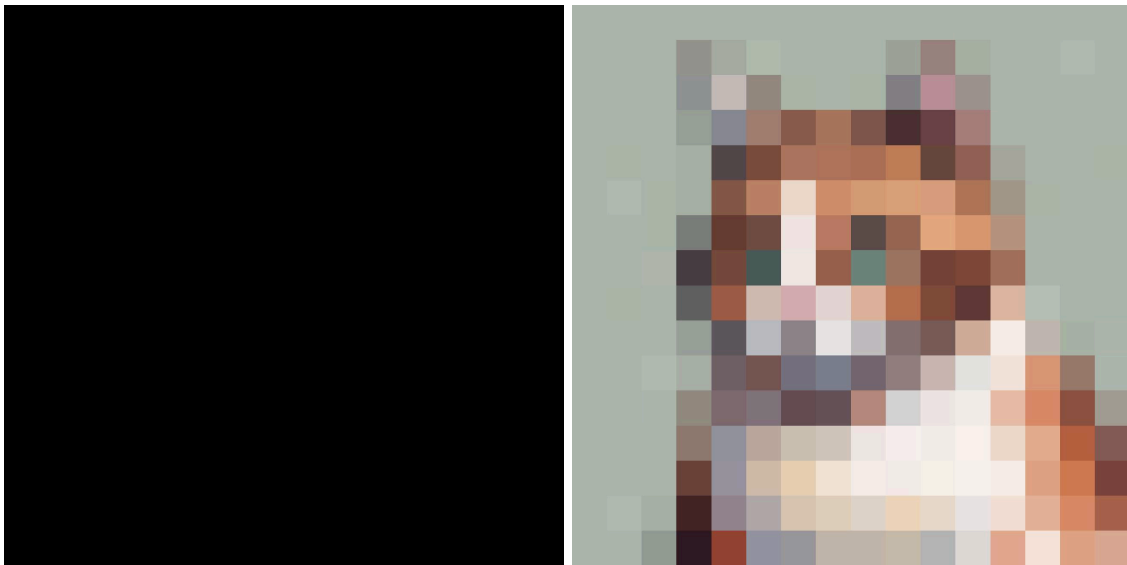
Masukkan target kompresi (0-1): 0.5

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test1.jpg

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test1.gif


```
-----  
                                Kompresi gambar selesai!  
-----  
Waktu eksekusi: 4.012 detik  
Ukuran sebelum: 372550 bytes  
Ukuran setelah: 55568 bytes  
Persentase kompresi: 85.08%  
Kedalaman pohon: 4  
Jumlah simpul: 333  
  
Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test1.jpg  
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test1.gif  
D:\Tucil2 Stima\test\output\test1.gif
```

Output penyelesaian program test1.jpg yang telah disimpan pada file test1.jpg dan test1.gif



4.2. Test Case 2

test2.png



Tampilan pertama saat program baru dijalankan

[illegible]

```
Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.  
Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.  
Masukkan threshold: 0.5  
  
Masukkan ukuran blok minimum: 100  
  
Masukkan target kompresi (0-1): 0.5  
  
Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test2.png  
Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test2.gif
```

Output penyelesaian program test2.png pada terminal

```
-----  
                        Kompresi gambar selesai!  
-----  
Waktu eksekusi: 1.681 detik  
Ukuran sebelum: 404805 bytes  
Ukuran setelah: 32217 bytes  
Persentase kompresi: 92.04%  
Kedalaman pohon: 4  
Jumlah simpul: 305
```

Output penyelesaian program test2.png yang telah disimpan pada file test2.png dan test2.gif

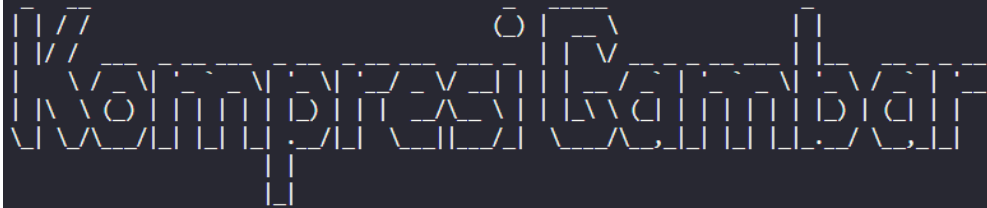


4.3. Test Case 3

test3.jpg



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test3.jpg

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

3

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 0.5

Masukkan ukuran blok minimum: 100

Masukkan target kompresi (0-1): 0.5

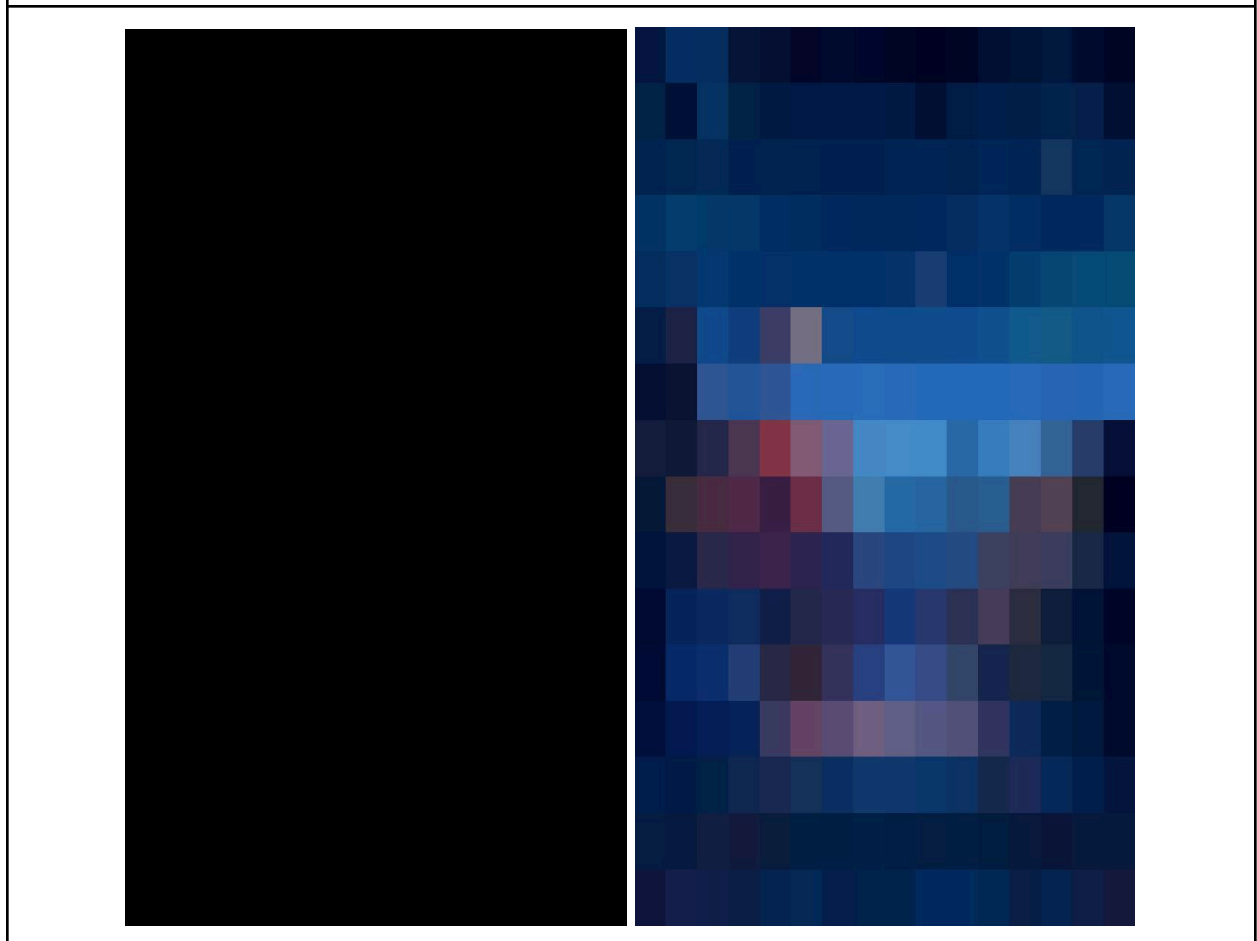
Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test3.jpeg

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test3.gif

Output penyelesaian program test3.jpg pada terminal

```
-----  
                                Kompresi gambar selesai!  
-----  
Waktu eksekusi: 2.196 detik  
Ukuran sebelum: 128453 bytes  
Ukuran setelah: 28835 bytes  
Persentase kompresi: 77.55%  
Kedalaman pohon: 4  
Jumlah simpul: 341  
  
Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test3.jpeg  
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test3.gif
```

Output penyelesaian program test3.jpg yang telah disimpan pada file test3.jpeg dan test3.gif



4.4. Test Case 4

test4.jpg



Tampilan pertama saat program baru dijalankan

```
-----  
KomputerLab  
-----  
Clarissa Nethania Tambunan / 13523016  
Shannon Aurellius Anastasya Lie / 13523019  
-----  
Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!  
Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test4.jpg  
Pilihan Metode Error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index Measure (SSIM)  
Masukkan pilihan (1/2/3/4/5):  
4  
Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.  
Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.  
Masukkan threshold: 0.5  
Masukkan ukuran blok minimum: 100  
Masukkan target kompresi (0-1): 0.5
```

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test4.png

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test4.gif

Output penyelesaian program test4.jpg pada terminal

```
-----  
                        Kompresi gambar selesai!  
-----  
Waktu eksekusi: 0.346 detik  
Ukuran sebelum: 10659 bytes  
Ukuran setelah: 1988 bytes  
Persentase kompresi: 81.35%  
Kedalaman pohon: 2  
Jumlah simpul: 21  
  
Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test4.png  
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test4.gif
```

Output penyelesaian program test4.jpg yang telah disimpan pada file test4.png dan test4.gif



4.5. Test Case 5

test5.jpg



Tampilan pertama saat program baru dijalankan

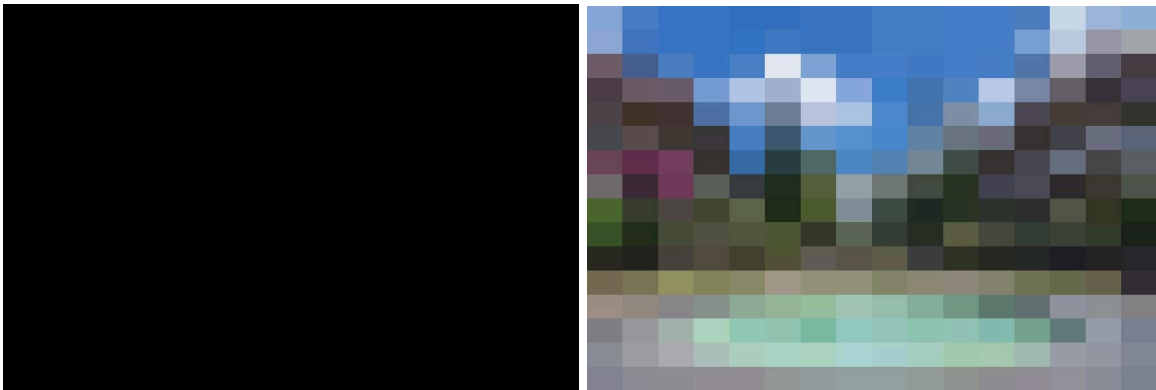
```
-----  
KOMPRESI GAMBAR MENGGUNAKAN METODE QUADTREE  
-----  
Clarissa Nethania Tambunan / 13523016  
Shannon Aurellius Anastasya Lie / 13523019  
-----  
Haiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!  
Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test5.jpg  
Pilihan Metode Error:  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
5. Structural Similarity Index Measure (SSIM)  
Masukkan pilihan (1/2/3/4/5):  
5  
Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.  
Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.  
Masukkan threshold: 0.5  
Masukkan ukuran blok minimum: 100
```

```
Masukkan target kompresi (0-1): 0.5  
Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test5.jpg  
Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test5.gif
```

Output penyelesaian program test5.jpg pada terminal

```
-----  
                        Kompresi gambar selesai!  
-----  
Waktu eksekusi: 1.941 detik  
Ukuran sebelum: 166698 bytes  
Ukuran setelah: 32158 bytes  
Persentase kompresi: 80.71%  
Kedalaman pohon: 4  
Jumlah simpul: 325  
  
Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test5.jpg  
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test5.gif
```

Output penyelesaian program test5.jpg yang telah disimpan pada file test5.jpg dan test5.gif

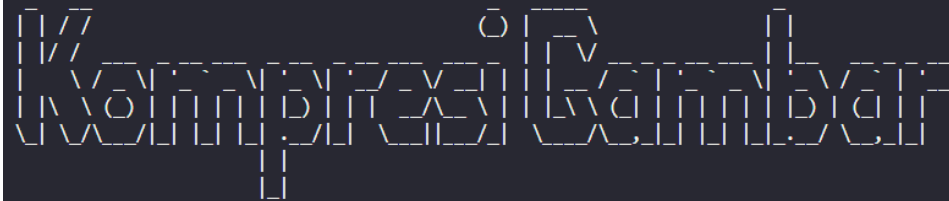


4.6. Test Case 6

test1.jpg



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test1.jpg

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

2

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 1

Masukkan ukuran blok minimum: 40

Masukkan target kompresi (0-1): 1

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test11.png

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test11.gif

Output penyelesaian program test1.jpg pada terminal

Kompresi gambar selesai!

Waktu eksekusi: 4.251 detik

Ukuran sebelum: 372550 bytes

Ukuran setelah: 165834 bytes

Persentase kompresi: 55.49%

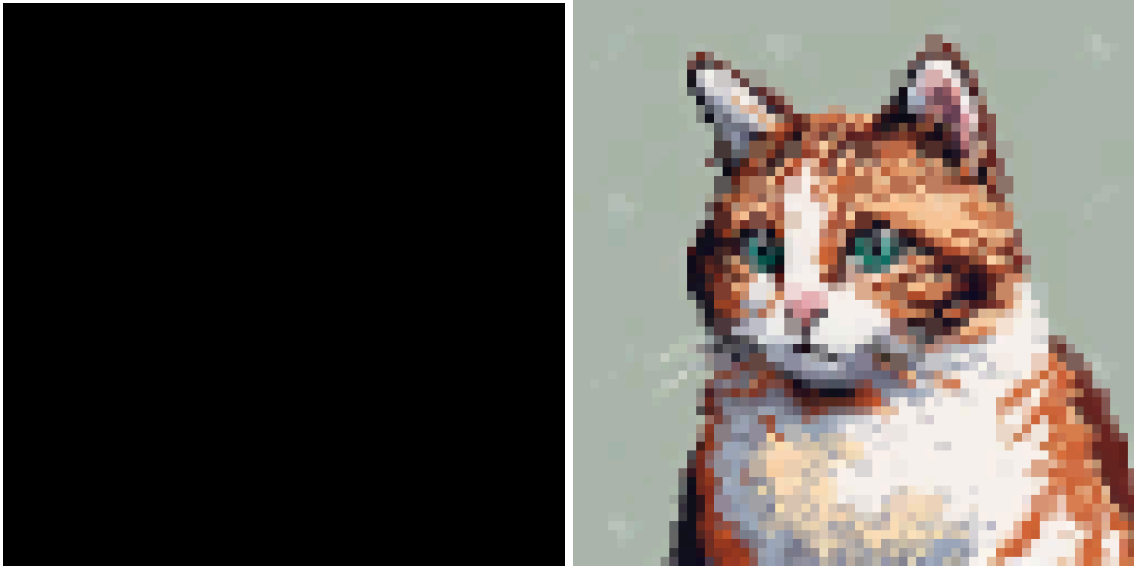
Kedalaman pohon: 6

Jumlah simpul: 3685

Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test11.png

GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test11.gif

Output penyelesaian program test1.jpg yang telah disimpan pada file test11.png dan test11.gif

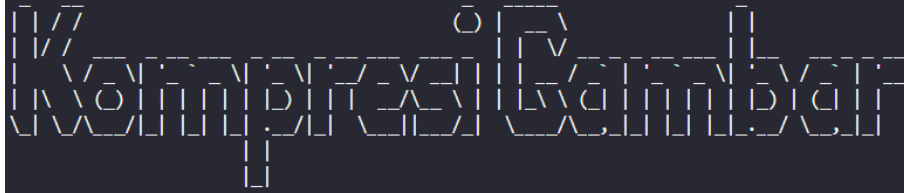


4.7. Test Case 7

test6.jpeg



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test6.jpeg

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

3

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 1

Masukkan ukuran blok minimum: 40

Masukkan target kompresi (0-1): 1

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test6.jpeg

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test6.gif

Output penyelesaian program test6.jpeg pada terminal

Kompresi gambar selesai!

Waktu eksekusi: 0.419 detik

Ukuran sebelum: 10335 bytes

Ukuran setelah: 3923 bytes

Persentase kompresi: 62.04%

Kedalaman pohon: 3

Jumlah simpul: 85

Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test6.jpeg

GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test6.gif

Output penyelesaian program test6.jpeg yang telah disimpan pada file test6.jpeg dan test6.gif

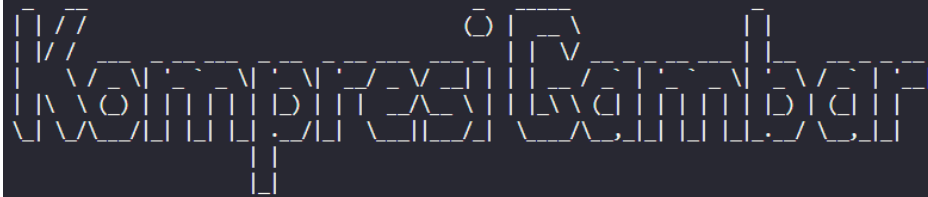


4.8. Test Case 8

test2.png



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test2.png

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

4

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 1

Masukkan ukuran blok minimum: 40

Masukkan target kompresi (0-1): 1

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test22.jpeg

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test22.gif

Output penyelesaian program test2.png pada terminal

Kompresi gambar selesai!

Waktu eksekusi: 2.108 detik

Ukuran sebelum: 404805 bytes

Ukuran setelah: 43467 bytes

Persentase kompresi: 89.26%

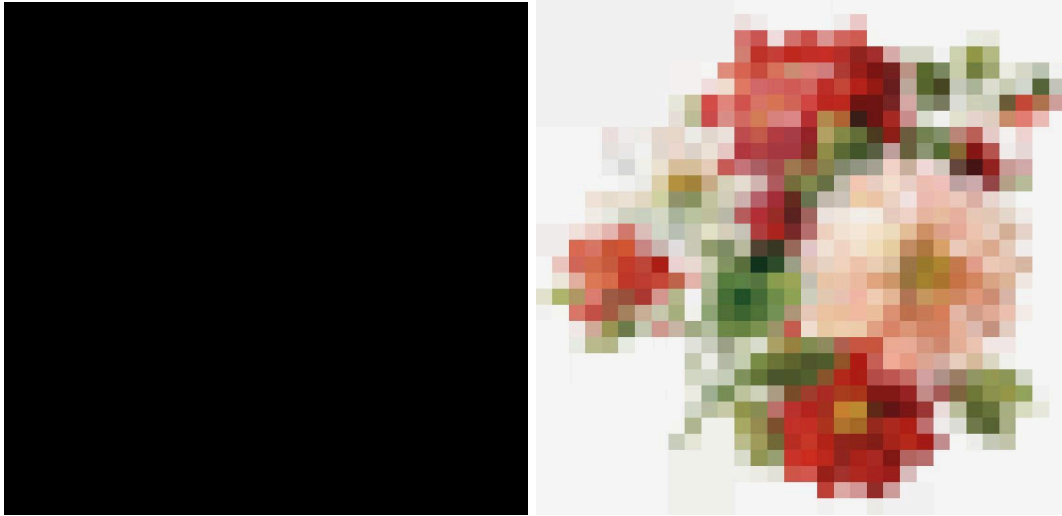
Kedalaman pohon: 5

Jumlah simpul: 969

Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test22.jpeg

GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test22.gif

Output penyelesaian program test2.png yang telah disimpan pada file test22.jpeg dan test22.gif

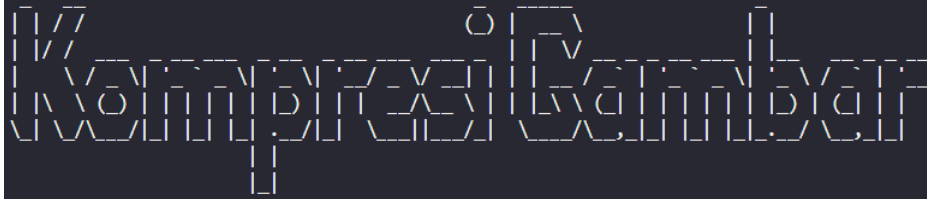


4.9. Test Case 9

test3.jpg



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\test2.png

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

5

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 0.5

Masukkan ukuran blok minimum: 40

Masukkan target kompresi (0-1): 1

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\test33.png

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\test33.gif

Output penyelesaian program test3.jpg pada terminal

Kompresi gambar selesai!

Waktu eksekusi: 2.056 detik

Ukuran sebelum: 404805 bytes

Ukuran setelah: 54146 bytes

Persentase kompresi: 86.62%

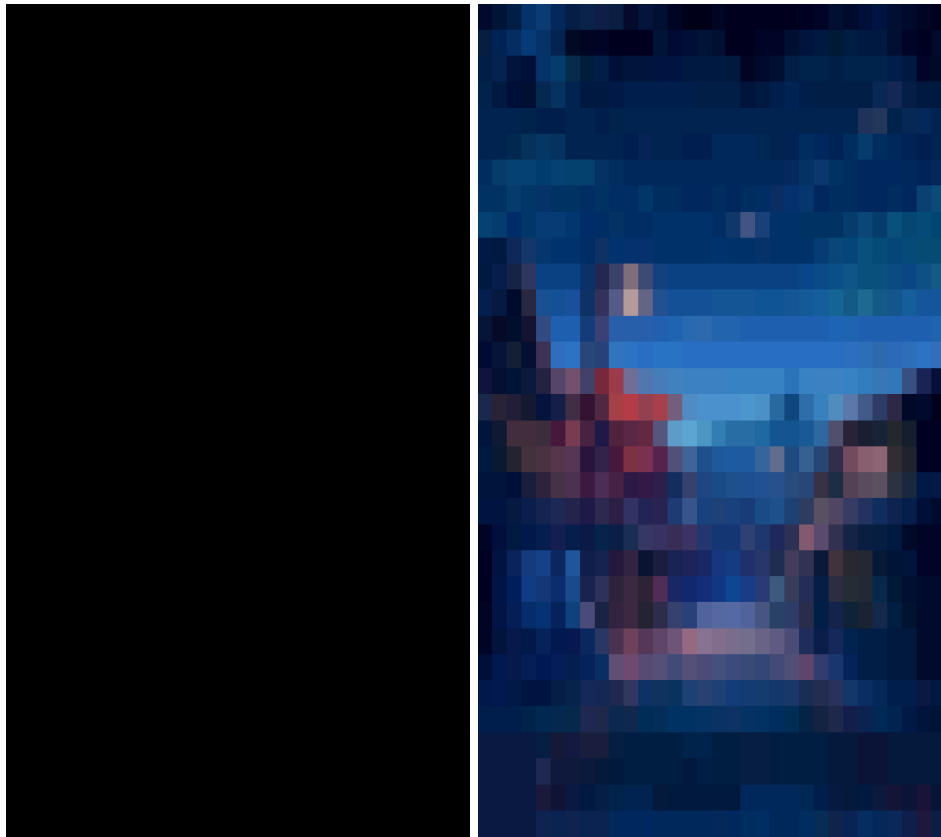
Kedalaman pohon: 5

Jumlah simpul: 1017

Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test33.png

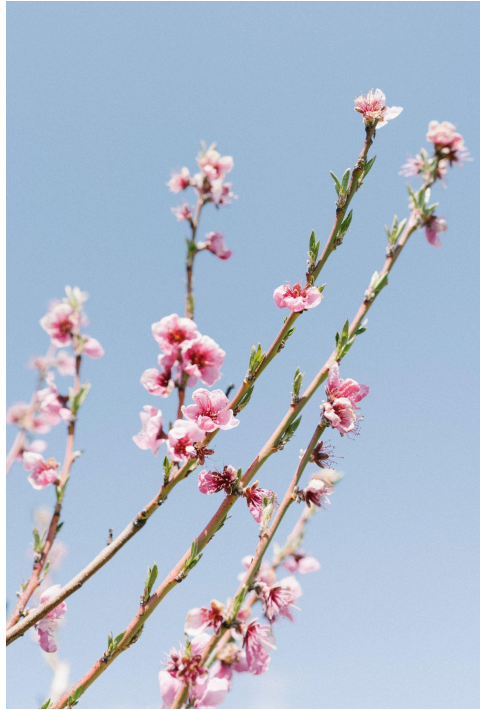
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\test33.gif

Output penyelesaian program test3.jpg yang telah disimpan pada file test33.png dan test33.gif

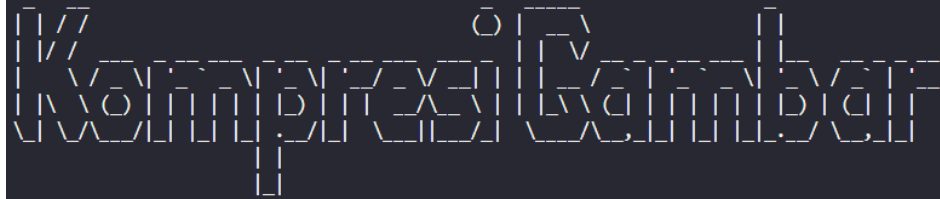


4.10. Test Case 10

flowers.jpg



Tampilan pertama saat program baru dijalankan



Clarissa Nethania Tambunan / 13523016
Shannon Aurellius Anastasya Lie / 13523019

Haiiiii, Selamat datang di program kompresi gambar menggunakan metode QuadTree!

Masukkan path gambar (absolut) input: D:\Tucil2 Stima\test\flowers.jpg

Pilihan Metode Error:

1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Masukkan pilihan (1/2/3/4/5):

1

Untuk pilihan metode 1, 4, dan 5, threshold harus antara 0 dan 1.

Untuk pilihan metode 2 dan 3, threshold harus antara 0 dan 255.

Masukkan threshold: 0.5

Masukkan ukuran blok minimum: 40

Masukkan target kompresi (0-1): 1

Masukkan path gambar (absolut) output: D:\Tucil2 Stima\test\output\flowers.jpeg

Masukkan path GIF (absolut) output: D:\Tucil2 Stima\test\output\flowers.gif

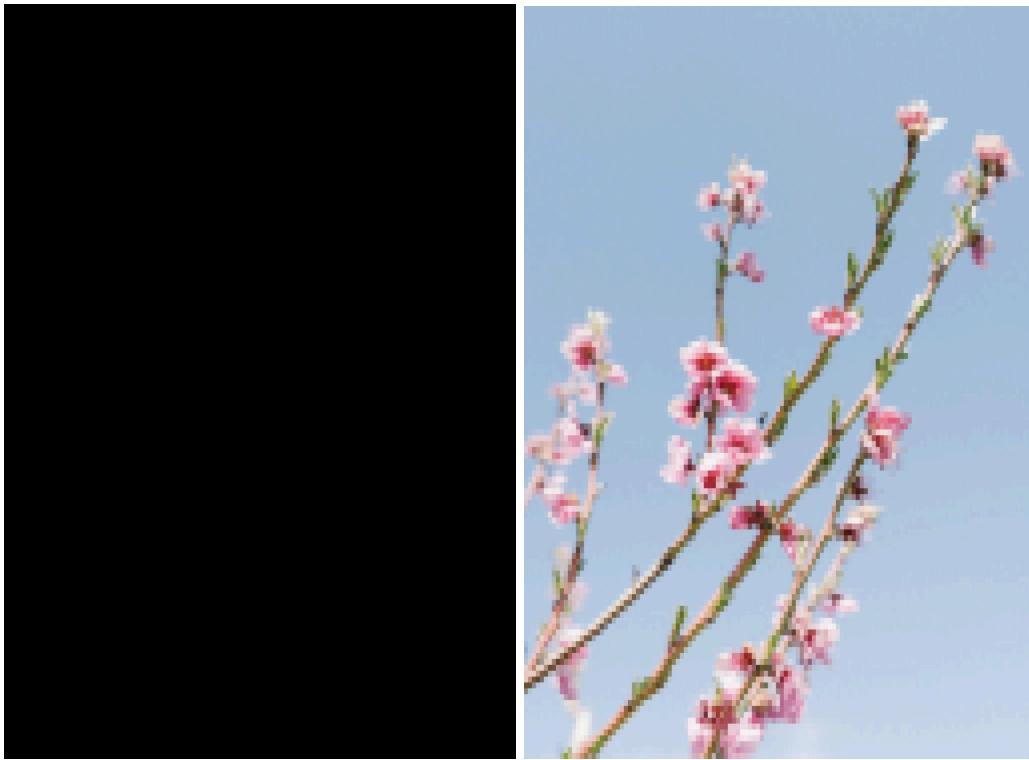
Output penyelesaian program flowers.jpg pada terminal

Kompresi gambar selesai!

Waktu eksekusi: 44.619 detik
Ukuran sebelum: 2667516 bytes
Ukuran setelah: 523931 bytes
Persentase kompresi: 80.36%
Kedalaman pohon: 7
Jumlah simpul: 21845

Gambar hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\flowers.jpeg
GIF hasil kompresi disimpan di: D:\Tucil2 Stima\test\output\flowers.gif

Output penyelesaian program flowers.jpg yang telah disimpan pada file flowers.jpeg dan flowers.gif



BAB V

KESIMPULAN DAN SARAN

5.1. Kesimpulan

Implementasi algoritma *divide and conquer* pada program kompresi gambar menggunakan metode *Quadtree* ini berhasil karena mampu mereduksi ukuran gambar tanpa menghilangkan struktur visual secara drastis berdasarkan masukan parameter pengguna. Program ini mampu membagi gambar menjadi blok-blok yang lebih kecil berdasarkan nilai error terhadap rata-rata warna, lalu menyatukan kembali hasilnya dalam bentuk gambar terkompresi dan animasi GIF yang merepresentasikan proses tersebut secara bertahap. Hasil kompresi ditampilkan melalui statistik seperti ukuran file sebelum dan sesudah, waktu eksekusi, kedalaman pohon, serta jumlah simpul. Hal ini menunjukkan efektivitas algoritma *Quadtree* dalam menangani kompresi gambar berbasis segmentasi spasial. Dalam hal kompleksitas, algoritma *Quadtree* memiliki kompleksitas waktu rata-rata $O(n \log n)$ untuk gambar dengan ukuran $n \times n$, tergantung pada seberapa sering pembagian kuadran dilakukan berdasarkan nilai error. Namun, pendekatan *divide and conquer* memungkinkan setiap kuadran diproses secara independen.

5.2. Saran

Untuk pengembangan selanjutnya, program ini dapat ditingkatkan lagi dari sisi kualitas kompresi dengan menambahkan metode perhitungan error lainnya yang lebih akurat agar pengguna dapat menilai sejauh mana hasil kompresi mempertahankan kualitas gambar asli. Selain itu, optimisasi algoritma pada proses rekursif juga bisa dipertimbangkan agar performa tetap efisien saat memproses gambar beresolusi tinggi. Visualisasi tambahan seperti pohon *quadtree* yang interaktif juga dapat menjadi nilai tambah dalam membantu memahami proses kompresi secara intuitif.

Tautan repository Github : https://github.com/4clarissaNT4/Tucil2_13523016_13523019

DAFTAR PUSTAKA

[1][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)

[2]<https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/Tucil2-Stima-2025.pdf>

LAMPIRAN

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	✓	