

Tugas Kecil II IF2211 Strategi Algoritma

Kompresi Gambar dengan Metode Quadtree



Disusun oleh:

Timothy Niels Ruslim (10123053)

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

DAFTAR ISI

DAFTAR ISI	2
1.1 Kompresi Gambar	3
1.2 Algoritma Divide and Conquer	3
1.3 Quadtree	4
BAB II	5
2.1 Algoritma Divide and Conquer dengan Quadtree untuk Kompresi Gambar	5
2.2 Metode Pengukuran Galat	7
2.2.1. Metode Pengukuran Galat Wajib	7
2.2.2. Metode SSIM	8
2.3 Binary Search untuk Mencari Target Kompresi	9
BAB III	10
3.1 Implementasi Algoritma Kompresi Gambar di Java	10
3.1.1. QuadtreeNode.java	10
3.1.2. Quadtree.java	11
3.1.3. ErrorCalculator.java	12
3.1.4. ThresholdCalculator.java	13
3.1.5. App.java (CLI Interface)	14
3.2 Source Code Program	15
3.2.1. QuadtreeNode.java	15
3.2.2. Quadtree.java	17
3.2.3. ErrorCalculator.java	19
3.2.4. ThresholdCalculator.java	22
3.2.5. App.java (CLI Interface)	23
BAB IV	29
4.1 Percobaan Program	29
4.1.1 Interface Masukan	29
4.1.2 Interface Keluaran	31
4.2 Pengujian Kompresi Gambar	32
4.2.1 Pengujian Berdasarkan Format File	33
4.2.2 Pengujian Berdasarkan Metode Perhitungan Galat	34
4.2.3 Pengujian Berdasarkan Target Persentase Kompresi	36
4.3 Analisis Kompleksitas Algoritma	39
DAFTAR PUSTAKA	40
LAMPIRAN	41

BAB I

DESKRIPSI MASALAH DAN LANDASAN TEORI

1.1 Kompresi Gambar

Kompresi gambar adalah suatu jenis kompresi data yang diterapkan pada gambar digital, yaitu dengan mengurangi data informasi gambar yang perlu disimpan file gambar. Permasalahan kompresi gambar penting untuk menghemat penyimpanan, sehingga dapat mempercepat transmisi dan *rendering* ataupun mengurangi penggunaan *bandwidth*.



Ada dua jenis kompresi gambar: *lossy* dan *lossless*. Kompresi *lossless* memungkinkan rekonstruksi ulang gambar orisinal tanpa kehilangan informasi, sedangkan kompresi *lossy* mereduksikan informasi yang disimpan misalnya menggunakan metode aproksimasi atau pembuangan data.

1.2 Algoritma *Divide and Conquer*

Algoritma *divide and conquer* adalah suatu algoritma yang memecah persoalan menjadi sub-persoalan, menyelesaiakannya secara rekursif, lalu menggabungkan solusi untuk menyelesaikan masalah asli. Perhatikan bahwa ini meliputi tiga tahap.

1. Proses *divide* yang membagi suatu persoalan menjadi beberapa sub-persoalan yang mirip dengan persoalan semula, namun memiliki ukuran yang lebih kecil.
2. Proses *conquer* yaitu penyelesaian setiap sub-persoalan, yang dapat dilakukan secara langsung jika sub-persoalan cukup kecil, atau secara rekursif dengan *divide and conquer* lagi jika masih berukuran cukup besar.
3. Proses *combine* yang menggabungkan solusi dari setiap sub-persoalan untuk membentuk solusi persoalan semula.

Karena sifat rekursifnya, skema umum dari algoritma *divide and conquer* dapat ditulis sebagai berikut.

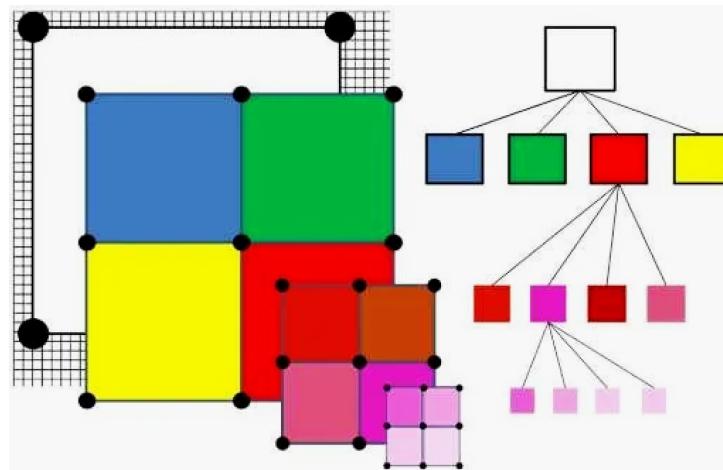
prosedur DivideAndConquer
Masukan: persoalan P , ukuran persoalan n
Luaran: solusi dari persoalan P

```
if  $n \leq n_0$  then
    SOLVE persoalan  $P$  yang berukuran  $n$ 
else
    DIVIDE menjadi  $r$  sub-persoalan,  $P_1, \dots, P_r$ , yang masing-masing berukuran  $n_1, \dots, n_r$ 
    for each  $P_1, \dots, P_r$  do
        DivideAndConquer( $P_i, n_i$ )
    endfor
    COMBINE solusi dari  $P_1, \dots, P_r$  menjadi solusi persoalan semula
endif
```

Dengan ini, terlihat bahwa algoritma *divide and conquer* bekerja paling baik untuk masukan atau instansi persoalan yang memiliki ukuran yang jelas, seperti larik, matriks, polinom, dan lainnya.

1.3 *Quadtree*

Quadtree adalah suatu data struktur pohon yang setiap simpulnya antara memiliki persis empat anak atau adalah daun. Dalam kata lain, dia adalah pohon yang bercabang empat.



Secara khusus, ada *region quadtree* yang menggunakan *quadtree* sebagai data struktur spasial, sehingga bermanfaat untuk mengorganisir ruang dua-dimensi. Suatu daerah, yang direpresentasikan sebagai simpul, dapat didekomposisikan menjadi empat kuadran, yaitu empat anak simpul. Kemudian, setiap kuadran dapat didekomposisikan lagi menjadi empat anak simpul secara rekursif.

BAB II

ALGORITMA KOMPRESI GAMBAR

2.1 Algoritma *Divide and Conquer* dengan *Quadtree* untuk Kompresi Gambar

Menggunakan data struktur *region quadtree*, dapat dirumuskan suatu algoritma *divide and conquer* yang dapat melakukan kompresi gambar *lossy*. Suatu gambar, yang dapat dianggap suatu daerah dua-dimensi, dapat direpresentasikan sebagai *quadtree*. Setiap simpul adalah suatu bagian dari gambar, dan secara teknis akan dikodifikasi dengan menyimpan koordinat pojok kiri-atas (x, y), ukuran (w, h), dan normalisasi warna (warna rataan) dari bagian relatif gambar orisinal. Di sini, kita akan bekerja dengan model RGB dari gambar. Jadi, rataan warna yang dimaksud adalah warna RGB dengan tiap kanal adalah rataan dari nilai kanal tersebut dari setiap piksel pada bagian.

Berikutnya, untuk melakukan kompresi gambar *lossy*, perlu dibangun *region quadtree* yang diinginkan dari gambar yang ingin dikompresi. Pertama, disimpan gambar asli sebagai simpul akar dari *quadtree*. Kemudian, lakukan algoritma *divide and conquer* seperti berikut secara rekursif.

1. *Divide*. Gambar bagian yang merepresentasikan suatu simpul akan dibagi menjadi empat kuadran bagian yang akan menjadi anak simpul. Pembagian akan dilakukan jika tiga kriteria berikut terpenuhi.
 - a. Distribusi warna pada gambar bagian “belum homogen”. Kualitas ini dapat dikuantifikasikan dengan mengecek apakah suatu statistik galat warna (yang menunjukkan perbedaan dari gambar bagian dengan warna rataannya) telah melewati suatu ambang batas yang telah ditetapkan. Jika galat terlalu besar relatif ambang batas, kita anggap “belum homogen”.
 - b. Ukuran bagian “cukup besar”. Lagi, secara praktis, ini berarti mengecek apakah ukuran luasnya lebih besar daripada suatu ukuran minimum yang telah ditetapkan. Jika luas bagian lebih besar dari ukuran minimum, bagian dianggap “cukup besar”.
 - c. Ukuran setiap kuadran dari bagian “cukup besar”. Karena pembagian menghasilkan empat kuadran, ini sama dengan mengecek apakah ukuran luas dibagi empat lebih besar daripada suatu ukuran minimum yang telah ditetapkan.

Perhatikan bahwa ketidakterpenuhinya kriteria tersebut akan menghasilkan suatu simpul daun.

2. *Conquer*. Setiap simpul yang bukan daun akan diterapkan kembali algoritma *divide and conquer* ini, yakni akan didekomposisikan menjadi empat kuadran lagi seterusnya secara rekursif. Simpul yang tidak memenuhi kriteria sehingga menjadi daun, akan diperlakukan sebagai suatu bagian satuan (*unit*), seperti pseudo-piksel, dengan hanya warna rataan bagian. Perhatikan bahwa proses ini menghilangkan informasi gambar asli. Ini karena bagian gambar yang direpresentasikan oleh daun tersebut hanya menyimpan bagian tersebut sebagai suatu warna rataan, bukan bagian-bagian lainnya hingga piksel terkecil. Ini yang menghasilkan kompresi gambar lossy yang diinginkan.
3. *Combine*. Setelah empat kuadran pada simpul anak suatu bagian gambar pada simpul telah diperoleh kompresinya, keempat bagian akan digabungkan kembali menjadi gambar yang utuh.

Langkah-langkah dari algoritma di atas dapat dituliskan sebagai *pseudocode* seperti berikut.

prosedur BuildQuadtree
Masukan
x : koordinat horizontal pojok kiri atas gambar bagian
y : koordinat horizontal pojok kiri atas gambar bagian
w : lebar gambar bagian
h : tinggi gambar bagian
Keluaran
node : simpul pada <i>region quadtree</i> yang merepresentasikan gambar bagian
Deklarasi
threshold : ambang batas untuk galat yang diterima
minSize : luas minimum bagian yang diterima
node = CreateNode (x, y, w, h)
avgColor \leftarrow warna rataan dari node
error \leftarrow galat dari node relatif warna rataan
if error > threshold and w \times h > minSize and w \times h / 2 > minSize then
w2 \leftarrow w / 2
h2 \leftarrow h / 2

```

buat BuildQuadtree(x, y, w2, h2) anak dari node
buat BuildQuadtree(x+w2, y, w2, h2) anak dari node
buat BuildQuadtree(x, y+h2, w2, h2) anak dari node
buat BuildQuadtree(x+w2, y+h2, w2, h2) anak dari node

endif

return node

```

2.2 Metode Pengukuran Galat

Untuk mengukur kehomogenan gambar bagian, dihitung suatu “galat” warna. Ada banyak metode yang dapat diimplementasikan sebagai berikut.

2.2.1. Metode Pengukuran Galat Wajib

Berikut adalah metode perhitungan galat wajib yang standar digunakan dalam pemrosesan gambar secara umum serta rumus dan jangkauan nilainya.

Metode	Rumus	Jangkauan
Variansi	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$ $\sigma_{\text{RGB}}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$	[0, 65025]
<i>Mean Absolute Difference</i> (MAD)	$\text{MAD}_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $ $\text{MAD}_{\text{RGB}} = \frac{\text{MAD}_R + \text{MAD}_G + \text{MAD}_B}{3}$	[0, 255]
<i>Max Pixel Difference</i>	$D_c = \max(P_{i,c}) - \min(P_{i,c})$ $D_{\text{RGB}} = \frac{D_R + D_G + D_B}{3}$	[0, 255]
Entropi	$H_c = - \sum_{i=1}^N \mathbb{P}_c(i) \log_2 (\mathbb{P}_c(i))$ $H_{\text{RGB}} = \frac{H_R + H_G + H_B}{3}$	[0, 8]

Perhatikan bahwa dicari galat pada setiap kanal RGB gambar bagian, lalu diambil rata-rata mereka sebagai galat dari seluruh gambar bagian. Juga, pengukuran galat di atas menyatakan kehomogenan jika nilainya *di bawah* suatu ambang batas.

2.2.2. Metode SSIM

Adapun suatu metode pengukuran galat yaitu *Structural Similarity Index Measure* (SSIM). Seperti yang namanya isyaratkan, dia adalah indeks yang mengukur seberapa mirip dua gambar. Rumus untuk menghitung SSIM untuk suatu kanal RGB antara dua gambar x dan y adalah

$$\text{SSIM}_c(\mathbf{x}, \mathbf{y}) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)},$$

dengan $C_1 = (K_1 L)^2$ dan $C_2 = (K_2 L)^2$, serta K_1 dan K_2 adalah parameter yang sangat kecil dan L adalah jangkauan dinamik dari nilai-nilai piksel (255 untuk kanal RGB dengan 8 bit). Untuk aplikasi umum, akan ditetapkan $K_1 = 0.01$ dan $K_2 = 0.03$.

Dalam kasus mengukur kehomogenan gambar untuk kompresi gambar algoritma yang telah dijelaskan, ini berarti akan dibandingkan suatu bagian gambar x dengan gambar monoton warna rataannya y . Perhatikan bahwa ini berarti

$$\mu_x = \mu_y \text{ dan } \sigma_x = \sigma_{xy} = 0.$$

Oleh karena itu, rumus SSIM dapat disederhanakan sebagai berikut.

$$\begin{aligned} \text{SSIM}_c(\mathbf{x}, \mathbf{y}) &= \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)} \\ &= \frac{(2\mu_x^2 + C_1)(C_2)}{(2\mu_x^2 + C_1)(\sigma_x^2 + C_2)} \\ &= \frac{C_2}{\sigma_x^2 + C_2} \\ &= \frac{1}{\sigma_x^2/C_2 + 1} \end{aligned}$$

Perhatikan bahwa SSIM sekarang menjadi berbalik terbalik, dengan translasi dan dilatasi, terhadap variansi gambar bagian. Lalu, jelas terlihat juga bahwa nilai SSIM untuk mengecek kehomogenan gambar hanya antara 0 dan 1. Terakhir, berbalik dengan metode pengukuran galat lainnya, SSIM menyatakan kehomogenan jika nilainya *di atas* suatu ambang batas, karena nilai yang lebih dekat ke 1 berarti gambar bagian lebih mirip atau *similar* dengan gambar monoton warna rataan.

Berikutnya, untuk SSIM, lebih diperuntungkan bahwa SSIM keseluruhan dari gambar bagian bukan hanya rataan SSIM dari setiap kanal, melainkan rataan berbobotnya. Karena mata manusia lebih sensitif terhadap warna hijau, dapat diberikan bobot yang lebih kepada kanal G. Penelitian telah menghasilkan standar berikut sebagai pembobotan terbaik.

$$\text{SSIM}_{\text{RGB}} = 0.299 \cdot \text{SSIM}_R + 0.587 \cdot \text{SSIM}_G + 0.114 \cdot \text{SSIM}_B$$

2.3 *Binary Search* untuk Mencari Target Kompresi

Adapun permasalahan menarik lainnya melibatkan kompresi gambar, yaitu untuk mencari nilai ambang batas, diberikan suatu ukuran bagian minimum, yang akan menghasilkan suatu persentase kompresi yang diinginkan. Untuk menyelesaikan masalah ini, dapat digunakan algoritma *decrease and conquer* berupa *binary search*.

Akan dimulai dengan melakukan tebakan nilai ambang batas. Tebakan awal dapat berupa masukan maupun titik tengah dari jangkauan nilai galat. Lalu, akan dilakukan kompresi gambar menggunakan ambang batas tersebut. Jika hasil persentase kompresi yang diperoleh belum sesuai dengan target, akan dinaikkan atau diturunkan tergantung metode pengukuran galat yang digunakan. Perubahan nilai ambang batas adalah dengan mencari titik tengah dari ambang batas yang ditebak dengan nilai maksimum atau nilai minimum. Proses ini dapat diulang secara rekursif sehingga tercapai persentase kompresi yang cukup dekat dengan target yang diinginkan.

Adapun bahwa, tergantung gambar awal dan ukuran minimum blok yang dipilih, bahwa ada target persentase kompresi yang tidak mungkin tercapai. Maka, *binary search* tidak akan pernah mencapai target tersebut. Untuk mencegah masalah ini, dapat dilakukan beberapa hal berikut.

- a. Pembatasan iterasi yang dilakukan oleh algoritma, sehingga rekursi sebaiknya diimplementasikan dengan loop daripada fungsi.
- b. Pengecekan apakah nilai maksimum dan minimum yang diperkecil sudah terlalu dekat.

Dengan ini, algoritma akan memberikan suatu nilai ambang batas yang akan menghasilkan nilai persentase kompresi sedekatnya dengan target, walaupun secara absolut tidak dekat sekali.

BAB III

IMPLEMENTASI DAN SOURCE CODE

3.1 Implementasi Algoritma Kompresi Gambar di Java

Akan diimplementasikan algoritma *divide and conquer* menggunakan *quadtree* untuk kompresi gambar dalam suatu program CLI di bahasa Java. Dipilih bahasa Java karena sistem *garbage collection* dan koleksi *library* di dalamnya yang akan membantu sekali dalam pemrosesan gambar. Berikut adalah modularisasi yang dilakukan.

3.1.1. QuadTreeNode.java

Pertama, dibuat terlebih dahulu kelas `QuadTreeNode` sebagai objek simpul dari *quadtree* yang merepresentasikan suatu bagian dari gambar yang ingin dikompresi. Objek ini memiliki beberapa atribut penting, beberapa sudah dianginkan dalam pembahasan algoritma, seperti berikut.

Atribut	Tipe Data	Deskripsi
patch	int[4]	Larik atau 4-tuple berbentuk (x, y, w, h) yang menyatakan koordinat pojok kiri atas bagian, juga dimensi horizontal dan vertikal bagian.
mode	int	Angka yang menyatakan jenis metode pengukuran galat yang digunakan.
color	Color	Objek warna <i>built-in</i> dari Java untuk warna rataan dari gambar bagian.
error	double	Galat, sesuai <code>mode</code> yang dipilih, dari gambar bagian.
depth	int	Kedalaman simpul pada <i>quadtree</i> , yaitu jumlah tepi antara akar dengan simpul.
children	ArrayList	Larik yang berisi semua simpul anak (yang antara berisi empat atau kosong jika daun).
leaf	boolean	Apakah simpul daun atau tidak.

Kemudian, `QuadTreeNode` memiliki beberapa metode yang membantu dalam pembentukan *quadtree* dan kompresi gambar seperti berikut.

Metode	Parameter	Luaran	Deskripsi
average	BufferedImage	Color	Fungsi yang menerima suatu objek gambar <i>built-in</i> Java, lalu mengembalikan warna rataannya.
compress	-	BufferedImage	Prosedur yang membuat gambar monoton warna rataan dengan ukuran sama dengan bagian gambar simpul.
divide	BufferedImage	-	Prosedur yang melakukan pembagian blok menjadi empat kuadran yang dijadikan empat simpul anak.

3.1.2. Quadtree.java

Berikutnya, dibuat objek pohon *region quadtree* dengan kelas `Quadtree`. Dia memiliki atribut-atribut pohon secara umumnya, tetapi juga gambar dan parameter kompresi.

Atribut	Tipe Data	Deskripsi
root	QuadtreeNode	Akar dari pohon, yaitu simpul untuk gambar orisinal yang ingin dikompresi.
treeDepth	int	Kedalaman pohon <i>quadtree</i> .
nodeCount	int	Jumlah simpul pada pohon <i>quadtree</i> .
image	BufferedImage	Gambar orisinal yang ingin dikompresi.
ERR_MODE	int	Angka yang menyatakan jenis metode pengukuran galat yang digunakan.
ERR_THRESHOLD	double	Angka yang menyatakan ambang batas untuk nilai galat saat pembagian.
MIN_SIZE	int	Ukuran minimum dari gambar bagian.

Kemudian, `Quadtree` memiliki beberapa metode yang mengimplementasikan algoritma *divide and conquer*, juga untuk membuat gambar hasil kompresi dan GIF proses kompresi.

Metode	Parameter	Luaran	Deskripsi
buildTree	QuadtreeNode	-	Prosedur yang menerapkan rekursi untuk algoritma <i>divide</i>

			<i>and conquer</i> untuk membangun <i>quadtree</i> sesuai parameter kompresi.
createImage	int	BufferedImage	Fungsi yang menghasilkan gambar hasil kompresi pada kedalam tertentu pada <i>quadtree</i> .
createImageRecursion	QuadtreeNode, int	BufferedImage	Fungsi yang melakukan teknis pembuatan, yaitu tahap <i>combine</i> , dari gambar hasil kompresi menggunakan rekursi.
renderImage	String, String ,	File	Fungsi yang mengekspor gambar hasil kompresi menjadi suatu file berdasarkan parameter alamat, nama, dan ekstensi file.
renderGif	String, String ,	File	Fungsi yang mengekspor GIF proses kompresi menjadi suatu file berdasarkan parameter alamat dan nama.

3.1.3. ErrorCalculator.java

Untuk perhitungan galat, digunakan kelas khusus `ErrorCalculator`. Dia hanya memiliki satu atribut yang sangat penting.

Atribut	Tipe Data	Deskripsi
errorModes	HashMap	Suatu <i>hashmap</i> yang memetakan angka metode perhitungan galat ke fungsi yang bersesuaian. Ini membuat kode lebih <i>maintainable</i> .

Lalu, `ErrorCalculator` memiliki banyak metode yang menerapkan setiap metode perhitungan galat yang telah dijelaskan sebelumnya.

Metode	Parameter	Luaran	Deskripsi
variance	BufferedImage ,	double	Fungsi yang menghitung variansi dari suatu gambar bagian yang direpresentasikan simpul.

mad	BufferedImage, QuadtreeNode	double	Fungsi yang menghitung <i>mean absolute deviation</i> (MAD) dari suatu gambar bagian yang direpresentasikan simpul.
mpd	BufferedImage, QuadtreeNode	double	Fungsi yang menghitung <i>max pixel difference</i> dari suatu gambar bagian yang direpresentasikan simpul.
entropy	BufferedImage, QuadtreeNode	double	Fungsi yang menghitung entropi dari suatu gambar bagian yang direpresentasikan simpul.
ssim	BufferedImage, QuadtreeNode	double	Fungsi yang menghitung <i>structural similarity index measure</i> (SSIM) dari suatu gambar bagian yang direpresentasikan simpul.

3.1.4. ThresholdCalculator.java

Adapun kelas `ThresholdCalculator` yang khusus dibuat untuk mencari ambang batas yang menghasilkan target persentase kompresi dengan mengimplementasikan algoritma *binary search* yang pernah dijelaskan sebelumnya.

Atribut	Tipe Data	Deskripsi
errorMax	HashMap	Suatu <i>hashmap</i> yang memetakan angka metode perhitungan galat ke nilai maksimum (pada jangkauannya) yang mungkin tercapai nilai galatnya.

Metode pada `ThresholdCalculator` adalah sebagai berikut.

Metode	Parameter	Luaran	Deskripsi
getThreshold	File, double, double, double, int, int	double	Fungsi yang menghitung ambang batas dengan <i>binary search</i> berdasarkan file gambar, target persentase kompresi, dan parameter kompresi yang diberikannya. Ini melibatkan pembangunan <i>quadtree</i> pada setiap tebakan ambang batas.

deleteAll	File	-	Prosedur yang digunakan untuk menghapus file <i>temporary</i> untuk setiap <i>quadtree</i> yang dibangun.
-----------	------	---	---

3.1.5. App.java (CLI Interface)

Program CLI yang mengintegrasikan algoritma kompresi gambar ini dengan interface ada pada `App.java`. Dia tidak ada atribut dan hanya memiliki dua metode.

Metode	Parameter	Luaran	Deskripsi
getInfo	File, string	String	Fungsi yang mencari antara nama file atau ekstensi file.
main	String []	-	Metode <i>entry point</i> program ini, yang mengurus masukan dan gambar, juga kompresi gambar melalui semua kelas yang telah dibuat sebelumnya.

Secara teknis, berikut cara kerja `main` di `App`. Pertama, program meminta masukan berupa hal-hal berikut.

1. Alamat dari gambar yang ingin dikompresi. Dapat berupa *alamat absolut* atau alamat relatif dari gambar. Bisa juga langsung menulis nama file jika gambar sudah ada pada folder `test\` pada directory projek (ini untuk memudahkan *testing* saja).
2. Metode perhitungan galat. Ini berupa bilangan bulat dari 1 hingga 5 berdasarkan *hashmap* yang dibuat di `ErrorCalculator`.
3. Ambang batas nilai galat untuk pembagian. Ini berupa bilangan riil positif yang ada pada jangkauan nilai galat berdasarkan metode perhitungan.
4. Ukuran minimum gambar bagian. Ini berupa bilangan bulat positif dan secara teknis adalah *luas* gambar bagian minimum.
5. (Opsional) Target persentase kompresi. Nilai persentase kompresi yang akan dicoba dicapai dengan *binary search*, dengan nilai 0 berarti menonaktifkan ini.
6. Alamat dari gambar hasil kompresi. Dapat berupa *alamat absolut*, alamat relatif, maupun `test` jika ingin mengakses folder `test\` pada directory projek untuk memudahkan saja. Perlu dicatat bahwa nama file gambar hasil kompresi adalah nama file asli, dengan ekstensi yang sama, tetapi dengan sufiks `_compressed`.

7. (Opsional) Alamat dari GIF proses kompresi. Dapat berupa *alamat absolut*, alamat relatif, maupun `test` jika ingin mengakses folder `test\` pada directory projek untuk memudahkan saja. Jika diisi dengan angka 0, GIF tidak akan dibuat. Perlu dicatat bahwa nama file GIF adalah nama file asli dengan sufiks `_compressed`.

Kemudian, program akan mencoba melakukan kompresi berdasarkan parameter dan gambar yang dimasukan oleh pengguna. Jika selesai, program akan memberikan keluaran berikut.

1. Waktu eksekusi kompresi (dalam ms).
2. Ukuran gambar sebelum kompresi (dalam MB).
3. Ukuran gambar sesudah kompresi (dalam MB).
4. Persentase kompresi yang tercapai.
5. Kedalaman pohon quadtree.
6. Banyak simpul pada pohon quadtree.
7. Gambar hasil kompresi pada alamat yang ditentukan.
8. (Opsional) GIF proses kompresi pada alamat yang ditentukan.

Terakhir, adapun bahwa program ini berjalan dalam suatu while loop, sehingga setelah program selesai mengeluarkan gambar hasil kompresi, dia akan mengulang meminta masukan pengguna untuk melakukan kompresi berikutnya.

3.2 Source Code Program

Berikut adalah *source code* dari program di bahasa Java dengan menerapkan semua yang telah dijelaskan di atas.

3.2.1. QuadtreeNode.java

```
import java.util.ArrayList;
import java.util.Arrays;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;

public class QuadtreeNode {

    // Image fields
    public int[] patch = new int[4]; // (x, y, w, h)
    public int mode;
    public Color color; // average color
    public double error;

    // Node fields
    public int depth;
    public ArrayList<QuadtreeNode> children; // {tl, tr, bl, br}
```

```

public boolean leaf = false;

// Constructor
public QuadtreeNode(BufferedImage image, int[] patch, int depth, int mode) {
    this.depth = depth;
    this.patch = patch;
    this.mode = mode;
    this.color = average(image);
    this.error = ErrorCalculator.calculateError(mode, image, this);
}

// Get average RGB color of image
public Color average(BufferedImage image) {
    double sumR = 0, sumG = 0, sumB = 0;
    int x = patch[0];
    int y = patch[1];
    int w = patch[2];
    int h = patch[3];
    int N = w * h;

    int[] pixels = new int[N];
    image.getRGB(x, y, w, h, pixels, 0, w);

    for (int i = 0; i < N; i++) {
        Color color = new Color(pixels[i]);
        sumR += color.getRed();
        sumG += color.getGreen();
        sumB += color.getBlue();
    }

    int R = (int) (sumR / N), G = (int) (sumG / N), B = (int) (sumB / N);
    return new Color(R, G, B);
}

// Creates monotone image of average color
public BufferedImage compress() {
    int w = patch[2];
    int h = patch[3];

    BufferedImage cImage = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    Graphics2D g = cImage.createGraphics();

    g.setColor(color);
    g.fillRect(0, 0, w, h);
    g.dispose();

    return cImage;
}

// Splits image into four sub-images
public void divide(BufferedImage image) {

    int x = patch[0];
    int y = patch[1];
    int w = patch[2];
    int h = patch[3];
    int mw = w / 2;
    int mh = h / 2;

    QuadtreeNode tl = new QuadtreeNode(image, new int[]{x, y, mw, mh}, depth +
}

```

```

    1, mode);
    QuadtreeNode tr = new QuadtreeNode(image, new int[]{x + mw, y, w - mw, mh},
    depth + 1, mode);
    QuadtreeNode bl = new QuadtreeNode(image, new int[]{x, y + mh, mw, h - mh},
    depth + 1, mode);
    QuadtreeNode br = new QuadtreeNode(image, new int[]{x + mw, y + mh, w - mw,
    h - mh}, depth + 1, mode);
    this.children = new ArrayList<>(Arrays.asList(tl, tr, bl, br));
}

}
}

```

3.2.2. Quadtree.java

```

import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.IOException;
import javax.imageio.ImageIO;
import javax.imageio.stream.FileImageOutputStream;
import javax.imageio.stream.ImageOutputStream;

public class Quadtree {

    // Tree Fields
    public QuadtreeNode root;
    public int treeDepth = 0;
    public int nodeCount = 0;

    // Compression Fields
    public BufferedImage image;
    public int ERR_MODE;
    public double ERR_THRESHOLD;
    public int MIN_SIZE;

    // Constructor
    public Quadtree(BufferedImage image, double ERR_THRESHOLD, int MIN_SIZE, int
ERR_MODE) {
        this.root = new QuadtreeNode(image, new int[]{0, 0, image.getWidth(),
image.getHeight()}, 0, ERR_MODE);
        this.image = image;
        this.ERR_MODE = ERR_MODE;
        this.ERR_THRESHOLD = ERR_THRESHOLD;
        this.MIN_SIZE = MIN_SIZE;

        buildTree(root);
    }

    // Build quadtree to compress image (divide)
    public void buildTree(QuadtreeNode node) {
        nodeCount++;

        // Make into leaf node if sufficient
        int size = node.patch[2] * node.patch[3];
        boolean withinThreshold = ERR_MODE == 5 ? node.error >= ERR_THRESHOLD :
node.error <= ERR_THRESHOLD;
        if (withinThreshold || size <= MIN_SIZE || size/4.0 < MIN_SIZE) {
            if (node.depth > treeDepth) treeDepth = node.depth;
            // System.out.println("Leaf: " + node.error);
            node.leaf = true;
            return;
        }
        ...
    }
}

```

```

    }

    // Recursion
    node.divide(image);
    // System.out.println("Branch: " + node.error);
    for (QuadtreeNode child : node.children) {
        buildTree(child);
    }
}

// Call recursion to create image
public BufferedImage createImage(int depth) {
    return createImageRecursion(root, depth);
}

// Create compressed image after building quadtree at given depth
private BufferedImage createImageRecursion(QuadtreeNode node, int depth) {

    if (node.leaf || node.depth == depth || node.children == null) {
        return node.compress();
    }

    int w = node.patch[2], h = node.patch[3];
    BufferedImage rImage = new BufferedImage(w, h, BufferedImage.TYPE_INT_RGB);
    Graphics2D g = rImage.createGraphics();

    BufferedImage tl = createImageRecursion(node.children.get(0), depth);
    BufferedImage tr = createImageRecursion(node.children.get(1), depth);
    BufferedImage bl = createImageRecursion(node.children.get(2), depth);
    BufferedImage br = createImageRecursion(node.children.get(3), depth);

    g.drawImage(tl, 0, 0, null);
    g.drawImage(tr, w/2, 0, null);
    g.drawImage(bl, 0, h/2, null);
    g.drawImage(br, w/2, h/2, null);
    g.dispose();

    return rImage;
}

// Instantly outputs file of fully compressed image
public File renderImage(String path, String fileName, String fileType) throws
IOException {

    File output = new File(path + "\\" + fileName + "_compressed." + fileType);
    BufferedImage compressedImage = createImage(treeDepth);
    ImageIO.write(compressedImage, fileType, output);

    return output;
}

// Outputs gif of full compression process
public File renderGif(String path, String fileName) throws
FileNotFoundException, IOException {

    File file = new File(path + "\\" + fileName + "_compressed.gif");
    ImageOutputStream output = new FileImageOutputStream(file);
    GifSequenceWriter writer = new GifSequenceWriter(
        output,
        BufferedImage.TYPE_INT_RGB,
        500,
        true
    );
}

```

```

        for (int i = 0; i <= treeDepth; i++) {
            BufferedImage image = createImage(i);
            writer.writeToSequence(image);
            if (i == treeDepth) for (int j = 0; j < 4; j++)
writer.writeToSequence(image);
        }

        writer.close();
        output.close();

        return file;
    }
}

```

3.2.3. ErrorCalculator.java

```

import java.util.HashMap;
import java.util.function.BiFunction;
import java.awt.Color;
import java.awt.image.BufferedImage;

public class ErrorCalculator {

    // Hashmap for error calculation mode
    public static final HashMap<Integer, BiFunction<BufferedImage, QuadtreeNode,
Double>> errorModes = new HashMap<>();
    static {
        errorModes.put(1, ErrorCalculator::variance);
        errorModes.put(2, ErrorCalculator::mad);
        errorModes.put(3, ErrorCalculator::mpd);
        errorModes.put(4, ErrorCalculator::entropy);
        errorModes.put(5, ErrorCalculator::ssim);
    }

    // Calculate the error based on the mode
    public static double calculateError(int mode, BufferedImage image,
QuadtreeNode node) {
        return errorModes.get(mode).apply(image, node);
    }

    // Get color Variance of image
    public static double variance(BufferedImage image, QuadtreeNode node) {
        double varR = 0, varG = 0, varB = 0;
        int x = node.patch[0];
        int y = node.patch[1];
        int w = node.patch[2];
        int h = node.patch[3];
        int N = w * h;

        int[] pixels = new int[N];
        image.getRGB(x, y, w, h, pixels, 0, w);

        int meanR = node.color.getRed(), meanG = node.color.getGreen(), meanB =
node.color.getBlue();
        for (int i = 0; i < N; i++) {
            Color pixelColor = new Color(pixels[i]);

            varR += Math.pow(pixelColor.getRed() - meanR, 2.0);
            varG += Math.pow(pixelColor.getGreen() - meanG, 2.0);
            varB += Math.pow(pixelColor.getBlue() - meanB, 2.0);
        }
    }
}

```

```

varR /= N;
varG /= N;
varB /= N;

    return (varR + varG + varB) / 3.0;
}

// Get color Mean Absolute Deviation (MAD) of image
public static double mad(BufferedImage image, QuadtreeNode node) {
    double madR = 0, madG = 0, madB = 0;
    int x = node.patch[0];
    int y = node.patch[1];
    int w = node.patch[2];
    int h = node.patch[3];
    int N = w * h;

    int[] pixels = new int[N];
    image.getRGB(x, y, w, h, pixels, 0, w);

    int meanR = node.color.getRed(), meanG = node.color.getGreen(), meanB =
node.color.getBlue();
    for (int i = 0; i < N; i++) {
        Color pixelColor = new Color(pixels[i]);

        madR += Math.abs(pixelColor.getRed() - meanR);
        madG += Math.abs(pixelColor.getGreen() - meanG);
        madB += Math.abs(pixelColor.getBlue() - meanB);
    }

    madR /= N;
    madG /= N;
    madB /= N;

    return (madR + madG + madB) / 3.0;
}

// Get color Max Pixel Difference (MPD) of image
public static double mpd(BufferedImage image, QuadtreeNode node) {
    int maxR = 0, maxG = 0, maxB = 0;
    int minR = 256, minG = 256, minB = 256;
    int x = node.patch[0];
    int y = node.patch[1];
    int w = node.patch[2];
    int h = node.patch[3];
    int N = w * h;

    int[] pixels = new int[N];
    image.getRGB(x, y, w, h, pixels, 0, w);

    for (int i = 0; i < N; i++) {
        Color pixelColor = new Color(pixels[i]);

        maxR = (pixelColor.getRed() > maxR) ? pixelColor.getRed() : maxR;
        maxG = (pixelColor.getGreen() > maxG) ? pixelColor.getGreen() : maxG;
        maxB = (pixelColor.getBlue() > maxB) ? pixelColor.getBlue() : maxB;

        minR = (pixelColor.getRed() < minR) ? pixelColor.getRed() : minR;
        minG = (pixelColor.getGreen() < minG) ? pixelColor.getGreen() : minG;
        minB = (pixelColor.getBlue() < minB) ? pixelColor.getBlue() : minB;
    }

    int dR = maxR - minR, dG = maxG - minG, dB = maxB - minB;
    return (dR + dG + dB) / 3.0;
}

// Get color Entropy of image

```

```

public static double entropy(BufferedImage image, QuadtreeNode node) {
    double[] histR = new double[256], histG = new double[256], histB = new
double[256];
    int x = node.patch[0];
    int y = node.patch[1];
    int w = node.patch[2];
    int h = node.patch[3];
    int N = w * h;

    int[] pixels = new int[N];
    image.getRGB(x, y, w, h, pixels, 0, w);

    for (int i = 0; i < N; i++) {
        Color pixelColor = new Color(pixels[i]);

        histR[pixelColor.getRed()]++;
        histG[pixelColor.getGreen()]++;
        histB[pixelColor.getBlue()]++;
    }

    double hR = 0, hG = 0, hB = 0;
    for (int i = 0; i < 256; i++) {
        histR[i] /= N;
        histG[i] /= N;
        histB[i] /= N;

        hR += (histR[i] > 0) ? histR[i] * (Math.log(histR[i]) / Math.log(2.0)) :
0;
        hG += (histG[i] > 0) ? histG[i] * (Math.log(histG[i]) / Math.log(2.0)) :
0;
        hB += (histB[i] > 0) ? histB[i] * (Math.log(histB[i]) / Math.log(2.0)) :
0;
    }

    return -(hR + hG + hB) / 3.0;
}

// Get color SSIM of image
public static double ssim(BufferedImage image, QuadtreeNode node) {
    double varR = 0, varG = 0, varB = 0;
    int x = node.patch[0];
    int y = node.patch[1];
    int w = node.patch[2];
    int h = node.patch[3];
    int N = w * h;

    int[] pixels = new int[N];
    image.getRGB(x, y, w, h, pixels, 0, w);

    int meanR = node.color.getRed(), meanG = node.color.getGreen(), meanB =
node.color.getBlue();
    for (int i = 0; i < N; i++) {
        Color pixelColor = new Color(pixels[i]);

        varR += Math.pow(pixelColor.getRed() - meanR, 2.0);
        varG += Math.pow(pixelColor.getGreen() - meanG, 2.0);
        varB += Math.pow(pixelColor.getBlue() - meanB, 2.0);
    }

    varR /= (double) N - 1.0;
    varG /= (double) N - 1.0;
    varB /= (double) N - 1.0;

    double C2 = Math.pow(0.03 * 255.0, 2);
    double siR = 1 / (varR / C2 + 1), siG = 1 / (varG / C2 + 1), siB = 1 /
(varB / C2 + 1);
}

```

```

        return 0.299 * siR + 0.587 * siG + 0.144 * siB;
    }

}

```

3.2.4. ThresholdCalculator.java

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;
import java.util.HashMap;
import javax.imageio.ImageIO;

public class ThresholdCalculator {

    // Hashmap for threshold range depending on mode
    public static final HashMap<Integer, Double> errorMax = new HashMap<>();
    static {
        errorMax.put(1, 65025.0);
        errorMax.put(2, 255.0);
        errorMax.put(3, 255.0);
        errorMax.put(4, 8.0);
        errorMax.put(5, 1.0);
    }

    // Find threshold given target compression percentage using BINARY SEARCH
    public static double getThreshold(File imageFile, double COMP_TARGET, double
ERR_THRESHOLD, int MIN_SIZE, int ERR_MODE) throws IOException {

        // Find image file
        String fileExtension = App.getInfo(imageFile, "extension");
        double originalSize = imageFile.length();

        // Create temporary directory to place images
        File tempDir = new File(System.getProperty("java.io.tmpdir") +
"/quadtree-temp");
        if (!tempDir.exists()) tempDir.mkdir();

        // Some parameters
        double high = errorMax.get(ERR_MODE);
        double low = 0.0;
        double threshold = ERR_THRESHOLD;
        int iteration = 0;
        int maxIterations = 10;

        // Binary search implementations
        while (iteration < maxIterations && high - low > 0.01) {

            if (iteration > 0) threshold = (high + low) / 2.0;

            // Create tree using given threshold
            BufferedImage image = ImageIO.read(imageFile);
            Quadtree tree = new Quadtree(image, threshold, MIN_SIZE, ERR_MODE);
            // System.out.println("Created tree");

            File compressedFile = tree.renderImage(tempDir.getAbsolutePath(),
"temp_" + iteration, fileExtension);
            // System.out.println("Created temporary image");

            // Check compression
            double compressedSize = compressedFile.length();
            double compressionPercent = 1.0 - (compressedSize / originalSize);
        }
    }
}

```

```

        // System.out.println("Iteration " + (iteration + 1) + ": using
threshold " + threshold + " to get " + (compressionPercent * 100) + "%
compression");
        if (Math.abs(compressionPercent - COMP_TARGET) < 0.005) {

            deleteAll(tempDir);
            return threshold; // found correct threshold

        } else if (compressionPercent > COMP_TARGET) { // too much compression

            if (ERR_MODE == 5) {
                low = threshold;
            } else {
                high = threshold;
            }

        } else { // too little compression

            if (ERR_MODE == 5) {
                high = threshold;
            } else {
                low = threshold;
            }

        }

        // Free up space
        compressedFile.delete();
        image.flush();
        tree = null;
        System.gc();

        iteration++;

    }

    deleteAll(tempDir);
    return (high + low) / 2.0;
}

// Delete all files in a directory (including itself)
public static void deleteAll(File file) {
    if (file.exists()) {
        if (file.isDirectory()) {
            for (File subfile : file.listFiles()) {
                deleteAll(subfile);
            }
        }
        file.delete();
        System.gc();
    }
}
}

```

3.2.5. App.java (CLI Interface)

```

import java.awt.image.BufferedImage;
import java.io.File;
import java.io.IOException;

import javax.imageio.ImageIO;
import java.text.DecimalFormat;
import java.util.Scanner;

```



```

        System.out.print("\nApakah gambar yang ingin dikompresi? [ketik alamat]
\u2192 ");
        String path = scanner.nextLine();
        File option1 = new File(path); // absolute path
        File option2 = new File("../test/" + path); // for ease of testing

        if (option1.exists() && option1.isFile()) {
            imageFile = option1;
        } else if (option2.exists() && option2.isFile()) {
            imageFile = option2;
        } else {
            System.out.println("File tidak ditemukan. Coba lagi.");
        }

    }

    // Input for error calculation method
    while (ERR_MODE == 0) {

        System.out.println("""
                            \nMetode perhitungan galat:
                            1. Variance
                            2. Mean Absolute Deviation (MAD)
                            3. Max Pixel Difference
                            4. Entropy
                            5. Structural Similarity Index Measure (SSIM)
                            """);
        System.out.print("Pilih metode perhitungan galat! [Ketik 1,2,...,5]
\u2192 ");
        if (scanner.hasNextInt()) { // check integer
            int mode = scanner.nextInt();
            if (mode >= 1 && mode <= 5) { // check valid mode
                ERR_MODE = mode;
            } else {
                System.out.println("Angka tidak valid. Coba lagi.");
            }
        } else {
            scanner.next();
            System.out.println("Angka tidak valid. Coba lagi.");
        }

        scanner.nextLine();
    }

    // Input for threshold
    while (ERR_THRESHOLD == -1) {

        int maxThreshold =
ThresholdCalculator.errorMax.get(ERR_MODE).intValue();
        System.out.print("\nMasukkan ambang batas! [Ketik 0-" + maxThreshold
+ "] \u2192 ");

        if (scanner.hasNextDouble()) { // check double
            double threshold = scanner.nextDouble();
            if (threshold >= 0 && threshold <= maxThreshold) { // check valid
threshold
                ERR_THRESHOLD = threshold;
            } else {
                System.out.println("Angka tidak valid. Coba lagi.");
            }
        } else {
            scanner.next();
            System.out.println("Angka tidak valid. Coba lagi.");
        }
    }
}

```

```

        scanner.nextLine();

    }

    // Input for minimum size
    while (MIN_SIZE == 0) {

        System.out.print("\nMasukkan ukuran blok minimum! [Ketik 1,2,...]
\u2192 ");

        if (scanner.hasNextInt()) { // check int
            int size = scanner.nextInt();
            if (size >= 1) { // check valid size
                MIN_SIZE = size;
            } else {
                System.out.println("Angka tidak valid. Coba lagi.");
            }
        } else {
            scanner.next();
            System.out.println("Angka tidak valid. Coba lagi.");
        }

        scanner.nextLine();
    }

    // Input for target compression percentage
    while (COMP_TARGET == -1) {

        System.out.print("\nMasukkan target persentase kompresi! [Ketik 0-1,
dengan 0 untuk menonaktifkan] \u2192 ");

        if (scanner.hasNextDouble()) { // check int
            double target = scanner.nextDouble();
            if (target >= 0 && target <= 1) { // check valid size
                COMP_TARGET = target;
            } else {
                System.out.println("Angka tidak valid. Coba lagi.");
            }
        } else {
            scanner.next();
            System.out.println("Angka tidak valid. Coba lagi.");
        }

        scanner.nextLine();
    }

    // Input for compressed image file path
    while (imagePath == null) {

        System.out.print("\nMau simpan hasil kompresi di mana? [Ketik alamat]
\u2192 ");

        String path = scanner.nextLine();
        File folder = new File(path); // absolute path

        if (folder.exists() && folder.isDirectory()) {
            imagePath = path;
        } else if (path.equals("test")) {
            imagePath = "../test";
        } else {
            System.out.println("Alamat tidak ditemukan. Coba lagi. ");
        }
    }
}

```

```

// Input for gif file path
while (gifPath == null) {

    System.out.print("\nMau simpan GIF di mana? [Ketik alamat, dengan 0
untuk menonaktifkan] \u2192 ");
    String path = scanner.nextLine();
    File option = new File(path); // absolute path

    if (path.equals("0")) {
        break;
    } else if (option.exists() && option.isDirectory()) {
        gifPath = path;
    } else if (path.equals("test")) {
        gifPath = "../test";
    } else {
        System.out.println("Alamat tidak ditemukan. Coba lagi. ");
    }
}

// Compression Process

System.out.println("\n=====");
long start = System.nanoTime();

BufferedImage image = ImageIO.read(imageFile);

// Find threshold value for target compression rate
if (COMP_TARGET != 0) {
    System.out.print("\nMencari ambang batas... ");
    ERR_THRESHOLD = ThresholdCalculator.getThreshold(imageFile,
COMP_TARGET, ERR_THRESHOLD, MIN_SIZE, ERR_MODE);
}

// Create quadtree for compression of image
System.out.println("\nGambar sedang dikompresi... ");
Quadtree tree = new Quadtree(image, ERR_THRESHOLD, MIN_SIZE, ERR_MODE);
File compressedFile = tree.renderImage(imagePath, getInfo(imageFile,
"name"), getInfo(imageFile, "extension"));

// Create compression process gif
if (gifPath != null) {
    System.out.println("GIF sedang dibuat... ");
    gifFile = tree.renderGif(gifPath, getInfo(imageFile, "name"));
}

long end = System.nanoTime();

System.out.println("\n=====");
// Output information
long time = (end - start) / 1000000;
double originalSize = imageFile.length() / (1024.0 * 1024.0);
double compressedSize = compressedFile.length() / (1024.0 * 1024.0);
double compressionPercent = (1.0 - (double) compressedFile.length() /
(double) imageFile.length()) * 100.0;
DecimalFormat df = new DecimalFormat("0.000");

System.out.println("\nWaktu kompresi : " + time + " ms ");
System.out.println("Ukuran sebelum : " +
df.format(originalSize) + " MB ");
System.out.println("Ukuran setelah : " +
df.format(compressedSize) + " MB ");

```

```

        System.out.println("Percentasi kompresi : " +
df.format(compressionPercent) + "% ");
        System.out.println("Kedalaman pohon : " + tree.treeDepth + "
");
        System.out.println("Banyak simpul pada pohon : " + tree.nodeCount + "
");

        System.out.println("\nGambar hasil kompresi dapat ditemukan di \"" + compressedFile.getAbsolutePath() + "\". ");
        if (gifFile != null) System.out.println("GIF proses kompresi dapat ditemukan di \\" + gifFile.getAbsolutePath() + "\". ");

        // Reset settings
imageFile = gifFile = null;
ERR_MODE = MIN_SIZE = 0;
ERR_THRESHOLD = COMP_TARGET = -1;
imagePath = gifPath = null;
image.flush();
System.gc();

System.out.println("\n=====");
=====

}

scanner.close();

}

}

```

BAB IV

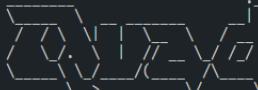
PENGUJIAN DAN ANALISIS

4.1 Percobaan Program

Berikutnya, akan dilakukan demonstrasi dan percobaan untuk *interface* dari program.

4.1.1 Interface Masukan

Berikut beberapa demonstrasi masukan program.

```
=====

=====
Made by Timothy Niels Ruslim (10123053)
=====
Apa gambar yang ingin dikompresi? [ketik alamat] → █
Landing terminal.

Apa gambar yang ingin dikompresi? [ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird.jpg
Metode perhitungan galat:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index Measure (SSIM)

Pilih metode perhitungan galat! [Ketik 1,2,...,5] → 1
Masukkan ambang batas! [Ketik 0-65025] → 69
Masukkan ukuran blok minimum! [Ketik 1,2,...] → 2
Masukkan target persentase kompresi! [Ketik 0-1, dengan 0 untuk menonaktifkan] → 0
Mau simpan hasil kompresi di mana? [Ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test
Mau simpan GIF di mana? [Ketik alamat, dengan 0 untuk menonaktifkan] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird.jpg

Masukan lengkap dan sukses.

Apa gambar yang ingin dikompresi? [ketik alamat] → bukan alamat
File tidak ditemukan. Coba lagi.
Apa gambar yang ingin dikompresi? [ketik alamat] → angka
File tidak ditemukan. Coba lagi.
Apa gambar yang ingin dikompresi? [ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\fileyangtidakada.jpg
File tidak ditemukan. Coba lagi.
Apa gambar yang ingin dikompresi? [ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird.jpg

Validasi alamat gambar asli.
```

Metode perhitungan galat:

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

Pilih metode perhitungan galat! [Ketik 1,2,...,5] → bukan angka
Angka tidak valid. Coba lagi.

Metode perhitungan galat:

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

Pilih metode perhitungan galat! [Ketik 1,2,...,5] → 0
Angka tidak valid. Coba lagi.

Metode perhitungan galat:

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

Pilih metode perhitungan galat! [Ketik 1,2,...,5] → 4

Validasi metode perhitungan galat.

Masukkan ambang batas! [Ketik 0-8] → bukan angka
Angka tidak valid. Coba lagi.

Masukkan ambang batas! [Ketik 0-8] → 9
Angka tidak valid. Coba lagi.

Masukkan ambang batas! [Ketik 0-8] → 4.2

Validasi ambang batas (berdasarkan metode).

Masukkan ukuran blok minimum! [Ketik 1,2,...] → bukan angka
Angka tidak valid. Coba lagi.

Masukkan ukuran blok minimum! [Ketik 1,2,...] → 0
Angka tidak valid. Coba lagi.

Masukkan ukuran blok minimum! [Ketik 1,2,...] → 6.9
Angka tidak valid. Coba lagi.

Masukkan ukuran blok minimum! [Ketik 1,2,...] → 1

Validasi ukuran blok minimum.

Masukkan target persentase kompresi! [Ketik 0-1, dengan 0 untuk menonaktifkan] → bukan angka
Angka tidak valid. Coba lagi.

Masukkan target persentase kompresi! [Ketik 0-1, dengan 0 untuk menonaktifkan] → -1
Angka tidak valid. Coba lagi.

Masukkan target persentase kompresi! [Ketik 0-1, dengan 0 untuk menonaktifkan] → 2
Angka tidak valid. Coba lagi.

Masukkan target persentase kompresi! [Ketik 0-1, dengan 0 untuk menonaktifkan] → 0

Validasi target persentase kompresi.

```
Mau simpan hasil kompresi di mana? [Ketik alamat] → bukan alamat  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan hasil kompresi di mana? [Ketik alamat] → 420  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan hasil kompresi di mana? [Ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird.jpg  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan hasil kompresi di mana? [Ketik alamat] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test
```

Validasi alamat gambar hasil kompresi.

```
Mau simpan GIF di mana? [Ketik alamat, dengan 0 untuk menonaktifkan] → bukan alamat  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan GIF di mana? [Ketik alamat, dengan 0 untuk menonaktifkan] → 69  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan GIF di mana? [Ketik alamat, dengan 0 untuk menonaktifkan] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\boat.jpeg  
Alamat tidak ditemukan. Coba lagi.  
  
Mau simpan GIF di mana? [Ketik alamat, dengan 0 untuk menonaktifkan] → D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test
```

Validasi alamat GIF proses kompresi.

4.1.2 Interface Keluaran

Berikut beberapa demonstrasi keluaran program.

```
=====  
Gambar sedang dikompresi...  
=====  
Waktu kompresi      : 17640 ms  
Ukuran sebelum     : 4.927 MB  
Ukuran setelah    : 2.492 MB  
Persentasi kompresi : 49.41%  
Kedalaman pohon   : 11  
Banyak simpul pada pohon : 2286725  
  
Gambar hasil kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\shinkansen_compressed.jpg".  
=====
```

Keluaran lengkap hasil proses kompresi.

```
=====  
Gambar sedang dikompresi...  
GIF sedang dibuat...  
=====  
Waktu kompresi      : 38964 ms  
Ukuran sebelum     : 5.353 MB  
Ukuran setelah    : 1.562 MB  
Persentasi kompresi : 70.820%  
Kedalaman pohon   : 11  
Banyak simpul pada pohon : 1384257  
  
Gambar hasil kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird_compressed.jpg".  
GIF proses kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird_compressed.gif".  
=====
```

Keluaran lengkap dengan GIF.

```
=====
Mencari ambang batas...
Gambar sedang dikompresi...

=====
Waktu kompresi      : 354844 ms
Ukuran sebelum      : 3.388 MB
Ukuran setelah      : 1.183 MB
Persentasi kompresi   : 65.089%
Kedalaman pohon     : 12
Banyak simpul pada pohon : 19563201

Gambar hasil kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\boat_compressed.jpeg".
=====
```

Keluaran lengkap dengan target kompresi.

```
=====
Mencari ambang batas...
Gambar sedang dikompresi...
GIF sedang dibuat...

=====
Waktu kompresi      : 60699 ms
Ukuran sebelum      : 5.353 MB
Ukuran setelah      : 1.653 MB
Persentasi kompresi   : 69.113%
Kedalaman pohon     : 11
Banyak simpul pada pohon : 1470945

Gambar hasil kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird_compressed.jpg".
GIF proses kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\bird_compressed.gif".
=====
```

Keluaran lengkap dengan target kompresi dan GIF.

```
=====
Waktu kompresi      : 354844 ms
Ukuran sebelum      : 3.388 MB
Ukuran setelah      : 1.183 MB
Persentasi kompresi   : 65.089%
Kedalaman pohon     : 12
Banyak simpul pada pohon : 19563201

Gambar hasil kompresi dapat ditemukan di "D:\ITB\Semester 4\Strategi Algoritma\Tugas Kecil 2\test\boat_compressed.jpeg".
=====

Apa gambar yang ingin dikompresi? [ketik alamat] →
|
```

Kembali ke masukan setelah keluaran selesai.

```
=====
Mencari ambang batas...
Gambar sedang dikompresi...
GIF sedang dibuat...
=====
```

Indikator tahap (mencari ambang batas, kompresi gambar, atau buat GIF).

4.2 Pengujian Kompresi Gambar

Berikutnya, akan diuji algoritma kompresi gambarnya sendiri. Dicatat bahwa untuk setiap pengujian, dibuat juga file GIF yang dapat dilihat di repository (karena tidak tampak di file pdf ini).

4.2.1 Pengujian Berdasarkan Format File

Program dapat menerima file jpg, jpeg, maupun png. Berikut pengujinya dengan metode, ambang batas, dan ukuran minimum yang sama tetapi file ekstensi berbeda.



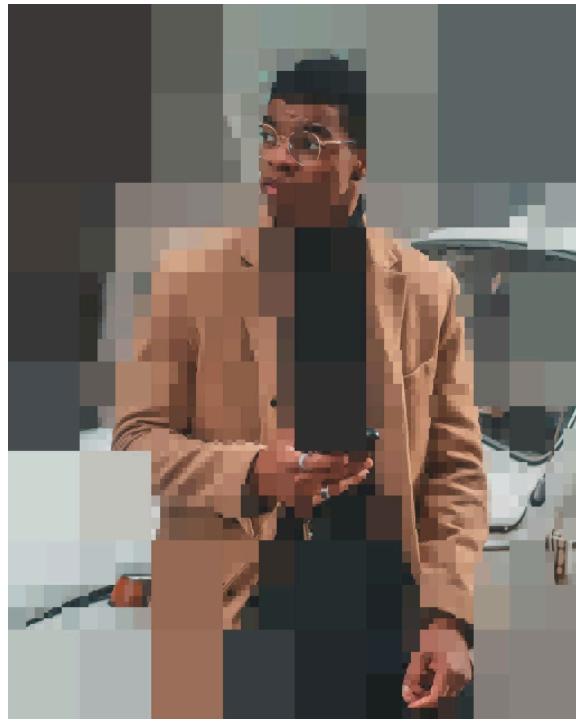
Gambar: shinkansen.jpg
Metode: entropy
Ambang batas: 6
Ukuran minimum: 4
Tanpa target kompresi



Waktu kompresi: 6174 ms
Ukuran sebelum: 4.927 MB
Ukuran setelah: 1.242 MB
Percentase: 74.789%
Kedalaman: 10
Banyak simpul: 73885



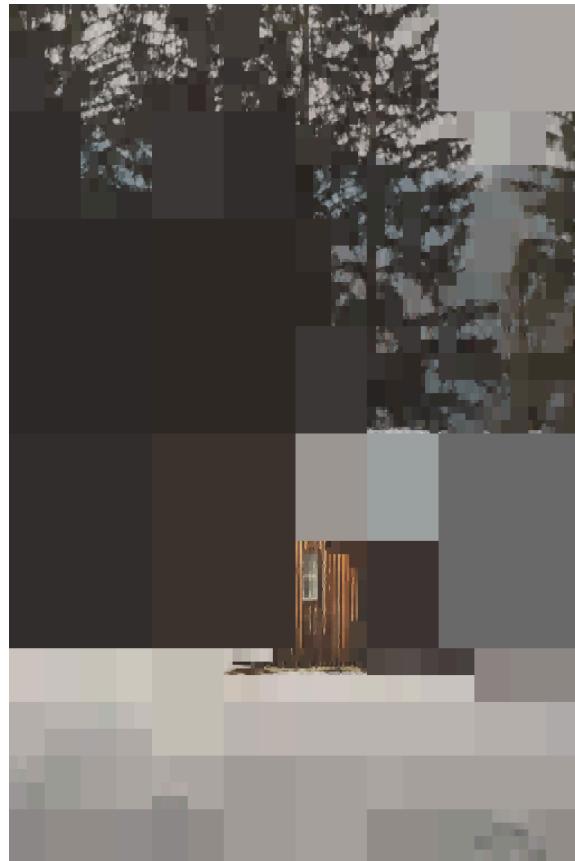
Gambar: suit.jpeg
Metode: entropy
Ambang batas: 6
Ukuran minimum: 4
Tanpa target kompresi



Waktu kompresi: 3708 ms
Ukuran sebelum: 5.144 MB
Ukuran setelah: 0.548 MB
Percentase: 89.345%
Kedalaman: 9
Banyak simpul: 10549



Gambar: cabin.png
Metode: entropy
Ambang batas: 6
Ukuran minimum: 4
Tanpa target kompresi



Waktu kompresi: 4717 ms
Ukuran sebelum: 11.856 MB
Ukuran setelah: 0.659 MB
Percentase: 94.438%
Kedalaman: 10
Banyak simpul: 18477

4.2.2 Pengujian Berdasarkan Metode Perhitungan Galat

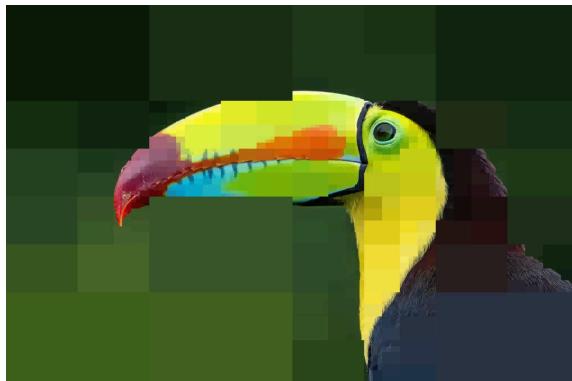
Adapun dapat digunakannya setiap metode perhitungan galat. Berikut pengujian dengan gambar dan ukuran minimum yang sama, tetapi variasi metode dan ambang batas.



Gambar: bird.jpg
Metode: variance



Waktu kompresi: 16933 ms
Ukuran sebelum: 5.353 MB

<p>Ambang batas: 200 Ukuran minimum: 1 Tanpa target kompresi</p>	<p>Ukuran setelah: 2.907 MB Persentase: 45.685% Kedalaman: 12 Banyak simpul: 6806153</p>
 <p>Gambar: bird.jpg Metode: mean absolute deviation Ambang batas: 20 Ukuran minimum: 1 Tanpa target kompresi</p>	 <p>Waktu kompresi: 2710 ms Ukuran sebelum: 5.353 MB Ukuran setelah: 0.452 MB Persentase: 91.559% Kedalaman: 12 Banyak simpul: 158901</p>
 <p>Gambar: bird.jpg Metode: max pixel difference Ambang batas: 100 Ukuran minimum: 1 Tanpa target kompresi</p>	 <p>Waktu kompresi: 5172 ms Ukuran sebelum: 5.353 MB Ukuran setelah: 0.977 MB Persentase: 81.744% Kedalaman: 12 Banyak simpul: 501897</p>



Gambar: bird.jpg
Metode: entropy
Ambang batas: 5.5
Ukuran minimum: 1
Tanpa target kompresi



Waktu kompresi: 5952 ms
Ukuran sebelum: 5.353 MB
Ukuran setelah: 0.757 MB
Percentase: 85.858%
Kedalaman: 10
Banyak simpul: 261849



Gambar: bird.jpg
Metode: SSIM
Ambang batas: 0.1
Ukuran minimum: 1
Tanpa target kompresi



Waktu kompresi: 3502 ms
Ukuran sebelum: 5.353 MB
Ukuran setelah: 0.615 MB
Percentase: 88.513%
Kedalaman: 12
Banyak simpul: 433637

4.2.3 Pengujian Berdasarkan Target Persentase Kompresi

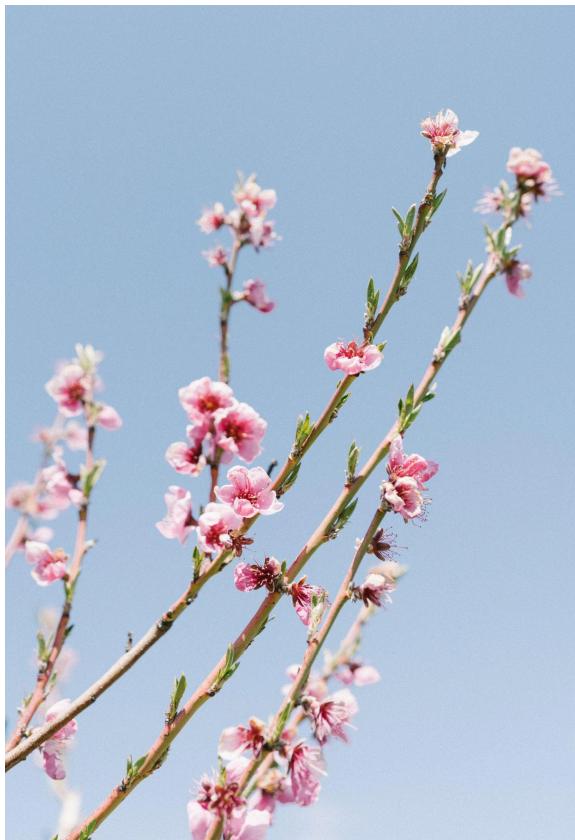
Berikut juga pengujian untuk kompresi dengan target persentase kompresi. Digunakan berbagai gambar dan parameter.



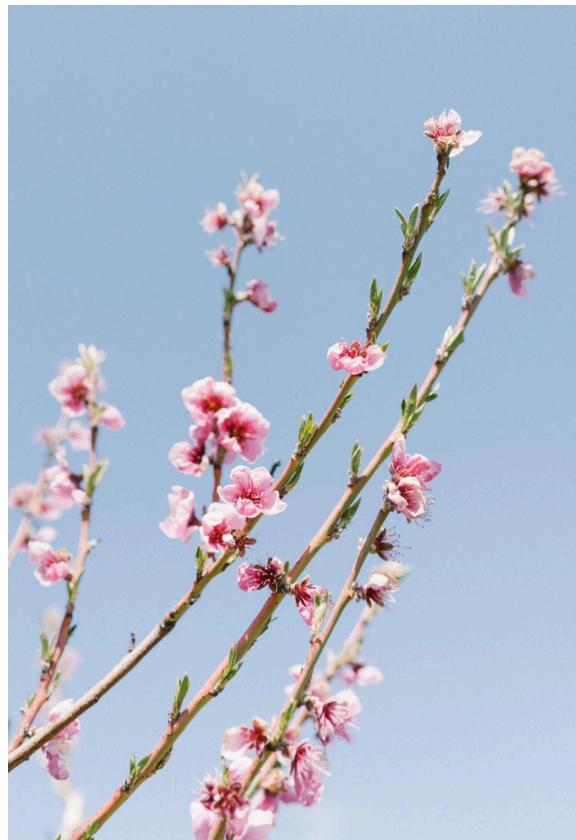
Gambar: boat.jpeg
Metode: variance
Ambang batas: 200
Ukuran minimum: 1
Target kompresi: 0.8



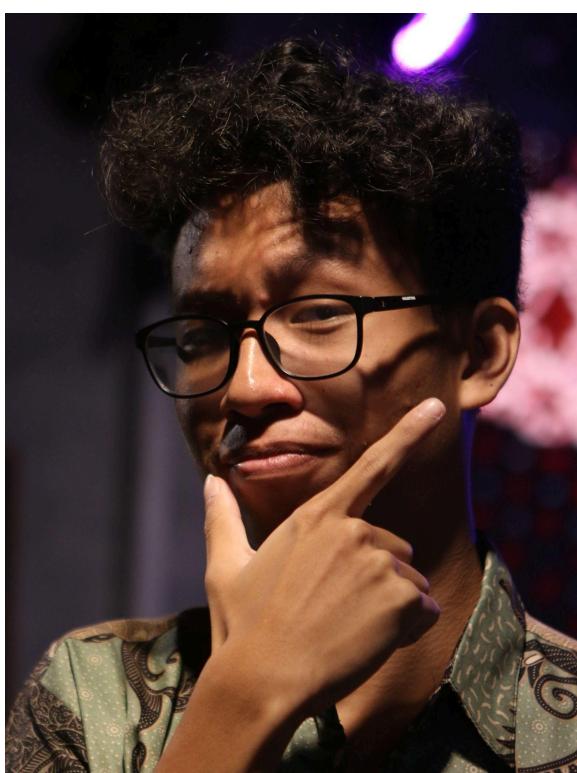
Waktu kompresi: 19002 ms
Ukuran sebelum: 3.388 MB
Ukuran setelah: 0.678 MB
Percentase: 79.978%
Kedalaman: 12
Banyak simpul: 292881



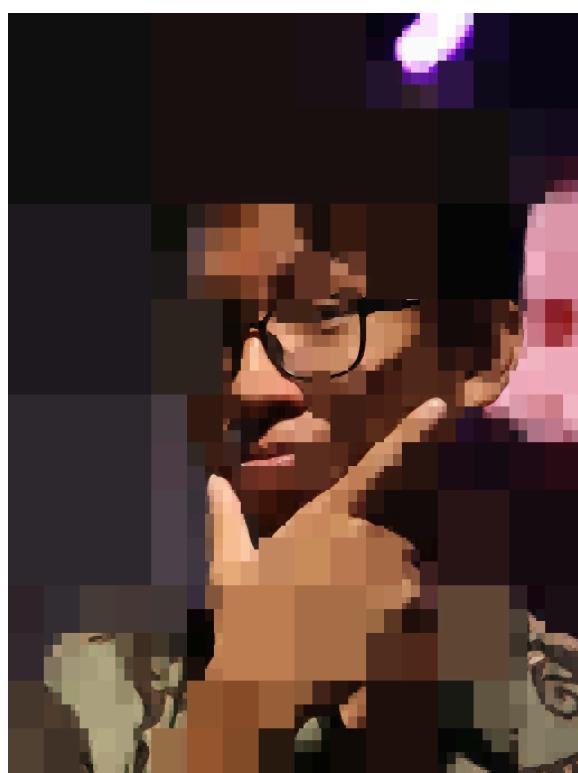
Gambar: flowers.jpg
Metode: entropy
Ambang batas: 4
Ukuran minimum: 1
Target kompresi: 0.8



Waktu kompresi: 50311 ms
Ukuran sebelum: 2.544 MB
Ukuran setelah: 0.789 MB
Percentase: 68.996%
Kedalaman: 11
Banyak simpul: 435513



Gambar: timo.jpg
Metode: SSIM
Ambang batas: 0.5
Ukuran minimum: 1
Target kompresi: 0.8



Waktu kompresi: 12531 ms
Ukuran sebelum: 5.278 MB
Ukuran setelah: 0.215 MB
Percentase: 95.935%
Kedalaman: 10
Banyak simpul: 6081



Gambar: tram.png



Waktu kompresi: 69037 ms

Metode: mean absolute difference Ambang batas: 30 Ukuran minimum: 1 Target kompresi: 0.73	Ukuran sebelum: 18.543 MB Ukuran setelah: 4.981 MB Persentase: 73.140% Kedalaman: 12 Banyak simpul: 1723785
--	---

4.3 Analisis Kompleksitas Algoritma

Berdasarkan pengujian yang dilakukan, terasa dampak ukuran dimensi gambar file yang dipilih terhadap waktu eksekusi kompresi. Gambar yang lebih besar juga umumnya terlihat menghasilkan pohon yang lebih dalam dengan lebih banyak simpul. Observasi ini akan diformalisasikan dengan analisis kompleksitas algoritma.

Misalkan $T(n)$ menyatakan kompleksitas waktu dari pembangunan *quadtree* lalu penggambungannya yang digunakan untuk melakukan kompresi gambar dengan dimensi $n \times n$ (untuk gambar tidak kotak, kompleksitas dapat dibatasi di atas dengan meninjau kompleksitas gambar berdimensi $\max\{m, n\} \times \max\{m, n\}$, sehingga analisis yang disederhanakan ini masih layak). Dalam source code, ini berkorespondensi dengan mencari kompleksitas dari fungsi `buildTree` dan `createImageRecursion`. Perhatikan bahwa dalam proses *divide*, gambar dibagi menjadi 4 sub-bagian dengan dimensi $n/2 \times n/2$. Lalu, dalam proses *conquer*, pembagian dilakukan lagi pada setiap sub-bagian. Akhirnya, dalam proses *combine*, karena perlu dibaca semua piksel dari setiap sub-bagian, dilakukan sekitar cn^2 operasi, dengan $c > 1$ suatu konstanta. Oleh karena itu, dapat ditulis

$$T(n) = 4T\left(\frac{n}{2}\right) + cn^2$$

adalah kompleksitas waktu algoritma. Perhatikan bahwa $a = 4 = 2^2 = b^d$, sehingga berdasarkan Teorema Master, dapat ditulis ulang kompleksitas algoritma sebagai

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

Hasil ini dapat dimengertikan secara intuitif. Untuk suatu gambar dengan dimensi $n \times n$, akan dibentuk suatu pohon *quadtree* dengan kedalaman maksimal sekitar $\log_2(n)$. Untuk setiap level, dilakukan pemrosesan sekitar n^2 piksel, karena dilakukannya perhitungan rataan, perhitungan galat, penggabungan gambar, dan lainnya. Oleh karena itu, ada sekitar $n^2 \log(n)$ operasi yang dilakukan.

DAFTAR PUSTAKA

- Britannica. "Data Compression | Lossless & Lossy Algorithms." *Encyclopedia Britannica*. Accessed April 8, 2025. Available at: www.britannica.com/technology/data-compression.
- ITU-R Recommendation BT.601-7. "Studio Encoding Parameters of Digital Television for Standard 4:3 and Wide-Screen 16:9 Aspect Ratios." *International Telecommunication Union (ITU)*, March 2011. Accessed April 9, 2025. Available at: www.itu.int/dms_pubrec/itu-r/rec/bt/r-rec-bt.601-7-201103-i!!pdf-e.pdf.
- IEEE Xplore Digital Library. "Document ID: 1284395." Accessed April 9, 2025. Available at: ieeexplore.ieee.org/document/1284395.
- Munir, Rinaldi. "07-Algoritma Divide and Conquer (2025) Bagian 1." *Informatika STEI ITB*. Accessed April 8, 2025. Available at: [informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf).
- Munir, Rinaldi. "08-Algoritma Divide and Conquer (2025) Bagian 2." *Informatika STEI ITB*. Accessed April 10, 2025. Available at: [informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](http://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf).
- OpenDSA Project. "PR Quadtrees." *OpenDSA CS3 Textbook*. Accessed April 8, 2025. Available at: opendsax.cs.vt.edu/ODSA/Books/CS3/html/PRquadtree.html.

LAMPIRAN

Berikut lampiran *checklist*.

No	Poin	Ya	Tidak
1	Program berhasil dikompilasi tanpa kesalahan.	✓	
2	Program berhasil dijalankan.	✓	
3	Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan.	✓	
4	Mengimplementasi seluruh metode perhitungan error wajib.	✓	
5	[Bonus] Implementasi persentase kompresi sebagai parameter tambahan.	✓	
6	[Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error.	✓	
7	[Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar.	✓	
8	Program dan laporan dibuat (kelompok) sendiri.	✓	

Berikut lampiran tautan.

Repository GitHub	github.com/timoruslim/Tucil2_10123053
-------------------	---