

LAPORAN TUGAS KECIL

IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:

Andrew Tedjapratama (13523148)

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

2025

DAFTAR ISI

DAFTAR ISI	1
BAB I.....	3
DESKRIPSI MASALAH.....	3
BAB II.....	4
SPESIFIKASI TUGAS	4
BAB III	6
TEORI SINGKAT.....	6
3.1 Struktur Data Quadtree	6
3.2 Algoritma Divide and Conquer	6
3.3 Metode Pengukuran Error	8
3.4 Parameter Tambahan dalam Kompresi	10
BAB IV	11
DESKRIPSI ALGORITMA.....	11
4.1 Inisialisasi data dan Struktur Data Quadtree	11
4.2 Proses Perhitungan Metode Error	13
4.3 Proses kompresi gambar dengan Algoritma Divide and Conquer	17
4.4 Penyelesaian dan Pengolahan Hasil	20
4.5 Kompleksitas Algoritma	20
BAB V	21
IMPLEMENTASI BONUS.....	21
5.1 Target Persentase Kompresi	21
5.2 Structural Similarity Index Measure (SSIM)	24
5.3 Output berupa GIF Visualisasi Proses.....	26
BAB VI	27
PENGETESAN PROGRAM	27
Test Case 1	27
Test Case 2	28
Test Case 3	29
Test Case 4	30
Test Case 5	31
Test Case 6	32
Test Case 7	33
BAB VII.....	34
KESIMPULAN DAN SARAN.....	34
7.1 Kesimpulan	34
7.2 Saran	34

BAB VIII	35
LAMPIRAN.....	35
Lampiran.....	35
Tabel Hasil	35

BAB I

DESKRIPSI MASALAH

Dalam dunia digital modern, kompresi gambar adalah salah satu aspek penting yang bermanfaat untuk penyimpanan dan transfer data visual yang efisien. Salah satu pendekatan yang seringkali dipakai adalah penggunaan struktur data Quadtree. Quadtree merupakan 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. Dengan hanya menyimpan informasi rata-rata warna pada area yang seragam di gambar, metode ini mampu mengurangi ukuran file tanpa perlu mengorbankan banyak detail visual penting.

Tugas kecil 1 dari mata kuliah Strategi Algoritma ini meminta sebuah penyelesaian masalah dengan membuat suatu program Quadtree Compressor menggunakan algoritma *divide and conquer* untuk mengimplementasikan kompresi gambar berbasis Quadtree. Program ini membaca gambar sebagai input, dan mengompresinya berdasarkan parameter tertentu, seperti metode pengukuran error, ambang batas variansi (*threshold*), ukuran blok minimum, dan target persentase kompresi. Program harus mampu memproses gambar dan menghasilkan gambar yang sudah terkompresi, beserta dengan informasi statistiknya sesuai parameter yang diinput oleh pengguna.

Algoritma *divide and conquer* diterapkan dengan membagi gambar menjadi empat blok, menghitung error pada setiap blok, dan membagi blok yang tidak seragam hingga mencapai kriteria tertentu. Program menghasilkan gambar terkompresi, informasi kompresi seperti waktu eksekusi, ukuran gambar sebelum dan sesudah kompresi, persentase kompresi, kedalaman pohon Quadtree (*depth*), dan jumlah simpul (*node*). Tugas ini juga menawarkan bonus implementasi seperti penyesuaian ambang batas otomatis untuk mencapai target persentase kompresi, penggunaan *Structural Similarity Index* (SSIM) sebagai metode pengukuran error, dan pembuatan visualisasi GIF dari proses pembentukan Quadtree.

BAB II

SPESIFIKASI TUGAS

- Buatlah program sederhana dalam bahasa C/C#/C++/Java (CLI) yang mengimplementasikan algoritma divide and conquer untuk melakukan kompresi gambar berbasis quadtree yang mengimplementasikan seluruh parameter yang telah disebutkan sebagai user input.
- Untuk lebih jelas, file input serta output berekstensi .png, .jpg, atau .jpeg saja.
- **Input:**

Program kompresi gambar berbasis Quadtree menerima sejumlah input dari pengguna, sebagai berikut:

1. Alamat gambar
Alamat absolut file gambar yang ingin dikompresi.
2. Metode perhitungan error
Pengguna dapat memilih salah satu metode pengukuran keseragaman blok dari metode-metode berikut:
 - Variance
 - Mean Absolute Deviation (MAD)
 - Max Pixel Difference
 - Entropy
 - Structural Similarity Index Measure (SSIM) (bonus)
3. Threshold (Ambang batas)
Nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
4. Minimum block size
Ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.
5. Target persentase kompresi
Target size image hasil kompresi yang ingin dicapai (berupa nilai *floating-point* antara 0 dan 1 (misalnya $0.7 = 70\%$). Jika bernilai 0, maka program tidak akan mengaktifkan mode penyesuaian threshold otomatis.
6. Alamat output gambar hasil kompresi
7. Alamat output GIF proses kompresi (*bonus*)

Alamat file untuk menyimpan visualisasi proses pembentukan Quadtree dalam format GIF.

- **Output Program**

Setelah proses kompresi selesai, program menghasilkan input sebagai berikut:

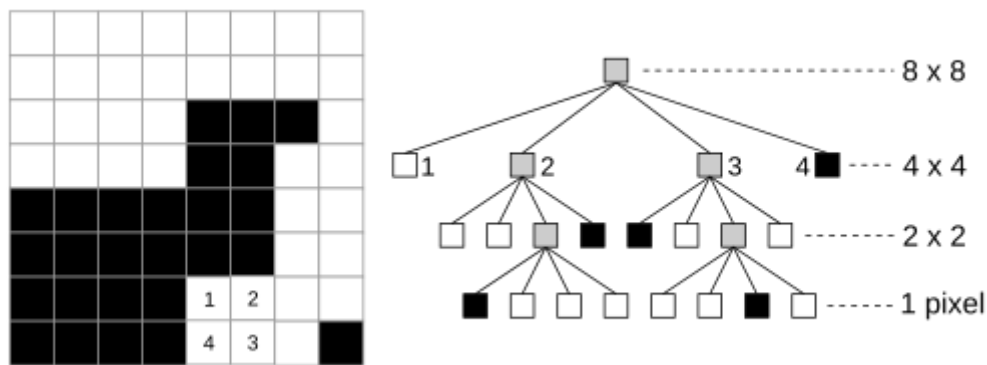
1. Waktu eksekusi
2. Ukuran gambar (*file size*) sebelum dan sesudah kompresi (dalam byte)
3. Persentase kompresi
4. Kedalaman pohon (*depth*)
5. Banyak simpul pada pohon (*nodes*)
6. Gambar hasil kompresi (*compressed image*) pada alamat output yang sudah ditentukan.
7. GIF hasil kompresi pada alamat yang sudah ditentukan (*bonus*)

BAB III

TEORI SINGKAT

3.1 Struktur Data Quadtree

Seperti yang sudah dijelaskan pada bab Deskripsi Masalah, proses kompresi menggunakan struktur data Quadtree dilakukan dengan 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.

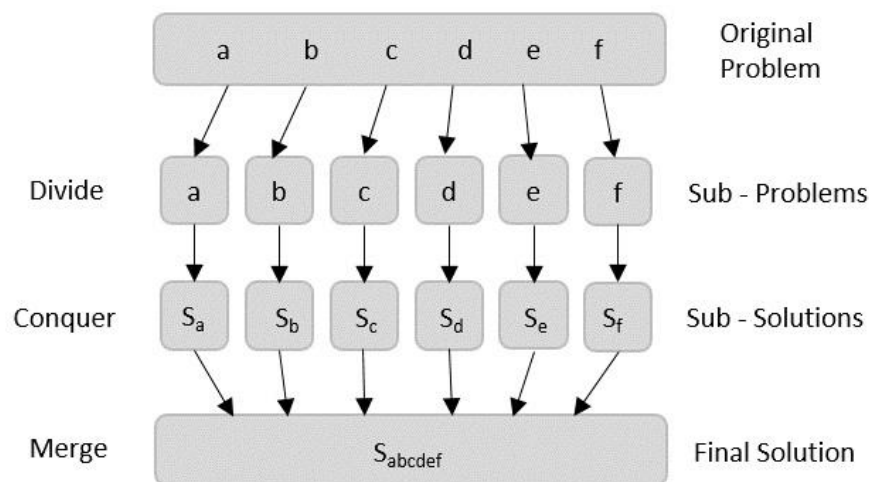


Dalam implementasi, sebuah Quadtree direpresentasikan sebagai simpul (*node*) dengan empat anak (*children*). Simpul daun (*leaf*) merepresentasikan area gambar yang dianggap seragam sehingga tidak perlu dibagi lagi, sementara simpul lain menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

3.2 Algoritma Divide and Conquer

Divide and conquer adalah pendekatan algoritmik yang bekerja dengan membagi suatu masalah besar menjadi beberapa submasalah lebih kecil, sehingga menyelesaikan submasalah

tersebut secara rekursif, dan menggabungkan hasilnya untuk memperoleh solusi akhir. Algoritma ini memiliki berbagai manfaat seperti efisiensinya bagi masalah yang bisa dibagi menjadi beberapa submasalah lebih kecil, kesederhanaan pendekatan yang memungkinkan pemahaman masalah lebih mudah, serta dapat dipakai berulang untuk masalah berbeda lainnya. Namun, tentunya pendekatan *divide and conquer* juga memiliki kelemahan dan kurang optimal pada masalah tertentu, seperti kompleksitas implementasi dan pemanfaatan memori yang lebih besar karena seringkali diimplementasi secara rekursif.



Gambar 3.2.1 Proses algoritma Divide and Conquer

(Sumber: https://www.tutorialspoint.com/data_structures_algorithms/divide_and_conquer.htm)

Pada kompresi gambar berbasis Quadtree, pendekatan ini diaplikasikan dengan membagi gambar menjadi 4 kuadran atau *node*, menganalisis masing-masing kuadran berdasarkan keseragaman warna, dan membaginya kembali jika belum memenuhi kriteria keseragaman. Kriteria dan syarat satu *node* akan terus membagi pada tugas ini ditentukan oleh variansi/error blok, ambang batas (*error*), dan ukuran block minimum (*minimum block size*).

Jika error di atas threshold, ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, maka blok tersebut akan dibagi menjadi empat sub-blok, dan dilanjutkan untuk setiap sub-blok secara rekursif. Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut dan blok tersebut akan dideklarasikan sebagai simpul daun (*leaf node*) yang tidak memiliki anak lagi.

3.3 Metode Pengukuran Error

Untuk setiap blok gambar yang sedang diproses, pengukuran error digunakan untuk menentukan tingkat keseragaman warna dalam suatu blok gambar. Nilai error dipakai untuk dibandingkan dengan threshold untuk memutuskan apakah blok gambar perlu dibagi lebih lanjut atau tidak. Berikut adalah beberapa metode pengukuran error yang digunakan dalam tugas ini:

a. Variance (σ^2)

Variance mengukur seberapa jauh nilai piksel dalam suatu blok menyimpang dari rata-ratanya. Variansi dihitung untuk setiap kanal warna (R, G, B), kemudian dirata-ratakan:

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

Di mana:

- σ^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
- N : Banyaknya piksel dalam satu blok

b. Mean Absolute Deviation (MAD)

MAD mengukur rata-rata selisih mutlak setiap piksel terhadap nilai rata-rata dalam blok, dihitung dengan formula berikut:

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

Di mana:

- 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
- N : Banyaknya piksel dalam satu blok

c. Max Pixel Difference

Metode ini mengukur selisih antara piksel maksimum dan minimum dalam satu kanal warna:

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

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

Di mana:

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 kanal warna c

d. Entropy

Entropi mengukur ketidakpastian distribusi nilai piksel. Entropi tinggi berarti piksel sangat bervariasi, dengan perhitungan sebagai berikut:

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

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

Di mana:

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)

e. Structural Similarity Index (SSIM) [Bonus]

SSIM adalah metode evaluasi kualitas visual yang mempertimbangkan struktur, kontras, dan luminansi gambar. Untuk kanal warna c, rumusnya:

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

Sementara, nilai SSIM total dihitung sebagai:

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

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.

3.4 Parameter Tambahan dalam Kompresi

Beberapa parameter penting yang digunakan dalam proses kompresi meliputi:

- **Threshold:** Nilai batas error yang digunakan untuk memutuskan pembagian blok.
- **Minimum Block Size:** Ukuran blok terkecil yang boleh dibagi.
- **Target Kompresi [BONUS]:** Persentase target pengurangan ukuran gambar, yang memicu penyesuaian threshold secara otomatis.
- **Persentase Kompresi:** Dihitung dengan rumus:

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

BAB IV

DESKRIPSI ALGORITMA

4.1 Inisialisasi data dan Struktur Data Quadtree

Program memulai dengan membaca dan memvalidasi data dari input file image berekstensi .png, .jpg, atau .jpeg (melalui class **ReadInput**) untuk menyimpan nilai-nilai input seperti image bertipe *BufferedImage* (*originalImage*), alamat file input dan output (*compressed*), metode error (*errorMethod*), ambang batas (*threshold*), ukuran blok minimum (*minBlockSize*), persentase target kompresi (*targetCompressionPercentage*), dan alamat output GIF (*gifPath*).

Data-data yang telah diinput tersebut kemudian disimpan dalam class **ImageInfo**. Selain itu, *ImageInfo* juga menyimpan *inputFormat* sebagai ekstensi file input (.png, .jpg, dan .jpeg) untuk penggunaan validasi dan sebagai parameter metode-metode file lainnya. Kelas ini dibuat dengan tujuan mempermudah penyimpanan dan pengaksesan semua informasi dan parameter kompresi di seluruh class lain.

```
1  class ImageInfo {
2      private String inputPath;
3      private String outputPath;
4      private int errorMethod;
5      private double threshold;
6      private int minBlockSize;
7      private BufferedImage originalImage;
8      private double targetCompressionPercentage;
9      private String gifPath;
10     private String inputFormat;
11
12
13     // ctor
14     public ImageInfo(BufferedImage image, String inputPath,
15                     String outputPath, int errorMethod, double threshold,
16                     int minBlockSize, double targetCompressionPercentage, String gifPath) {
17         this.inputPath = inputPath;
18         this.outputPath = outputPath;
19
20         this.errorMethod = errorMethod;
21         this.threshold = threshold;
22         this.minBlockSize = minBlockSize;
23
24         this.originalImage = image;
25         this.targetCompressionPercentage = targetCompressionPercentage;
26         this.gifPath = gifPath;
27
28         this.inputFormat = getExtension(new File(inputPath));
29     }
```

Gambar 4.1.1 Kelas ImageInfo

File utama untuk struktur data Quadtree terdapat pada kelas **QuadTreeNode**, yang berisi implementasi dari struktur data tersebut. Kelas ini menyimpan berbagai atribut, seperti koordinat titik node (x dan y) yang diinisialisasi dengan (0,0) atau atas kiri, lebar dan tinggi area yang direpresentasikan ($width$ dan $height$), *isLeaf* (boolean menandakan node adalah daun atau tidak memiliki anak), referensi ke empat anak node (*topLeft*, *topRight*, *bottomLeft*, *bottomRight*), dan nilai rata-rata komponen warna RGB untuk area yang di representasikan (*meanR*, *meanG*, *meanB*). Node pada program ini merepresentasikan blok dalam gambar.

Selain konstruktor Quadtree, terdapat metode **split()** yang berfungsi membagi node saat ini menjadi empat anak node dengan area yang sama besar. Metode split ini juga menghitung ukuran bagian masing-masing anak dan membuat empat node anak baru, serta menandai node sebagai *non-leaf*.

```
1  class QuadTreeNode {
2      private int x, y;
3      private int width;
4      private int height;
5      private boolean isLeaf;
6
7      private QuadTreeNode topLeft;
8      private QuadTreeNode topRight;
9      private QuadTreeNode bottomLeft;
10     private QuadTreeNode bottomRight;
11
12
13     private double meanR, meanG, meanB;
14     // ctor
15     public QuadTreeNode(int x, int y, int width, int height){
16         this.x = x;
17         this.y = y;
18         this.width = width;
19         this.height = height;
20         this.isLeaf = true;
21
22         this.topLeft = null;
23         this.topRight = null;
24         this.bottomLeft = null;
25         this.bottomRight = null;
26     }
27
28     public void split() {
29         int leftWidth = width / 2;
30         int rightWidth = width - leftWidth;
31         int topHeight = height / 2;
32         int bottomHeight = height - topHeight;
33
34         topLeft = new QuadTreeNode(x, y, leftWidth, topHeight);
35         topRight = new QuadTreeNode(x + leftWidth, y, rightWidth, topHeight);
36         bottomLeft = new QuadTreeNode(x, y + topHeight, leftWidth, bottomHeight);
37         bottomRight = new QuadTreeNode(x + leftWidth, y + topHeight, rightWidth, bottomHeight);
38         isLeaf = false;
39     }
```

Gambar 4.1.2 Kelas QuadTreeNode

4.2 Proses Perhitungan Metode Error

Perhitungan metode error dilakukan pada kelas **ErrorCalculation** dengan bantuan kumpulan utilitas dalam kelas **ChannelUtil**. Kelas ChannelUtil menangani operasi per channel (RGB) untuk memudahkan proses perhitungan error di kelas ErrorCalculation, dilakukan dengan memproses channel warna RGB dari gambar dan menyediakan fungsi untuk membantu perhitungan error.

Salah satu metode yang penting, meliputi *calculateAllMeans* yang berfungsi menghitung rata-rata nilai pixel untuk masing-masing channel (R, G, B) dalam sebuah blok, dan disimpan dalam array double. Ketiga channel warna dideklarasikan dalam integer konstan/final secara static. Metode tersebut akan digunakan untuk perhitungan error Variance, MAD, dan SSIM.

Kedua, terdapat metode *getChannelValues* yang berfungsi mengambil daftar nilai intensitas pixel (0-255) untuk satu channel dalam blok tertentu, yang dioutput dalam bentuk *list of integer* berisi average intensity values dari setiap channel R, G, B. Metode ini akan digunakan untuk menghitung Variance, MAD, Entropy, dan SSIM. Terakhir, method *getChannelMinMax* juga dibuat dengan mengembalikan nilai minimum dan maksimum pixel dalam sebuah blok untuk satu channel. Metode ini memiliki tujuan menghitung Max Pixel Difference (MPD).

```
1  class ChannelUtil {
2      public static final int redChannel = 0;
3      public static final int greenChannel = 1;
4      public static final int blueChannel = 2;
5
6      // * Calculate Mean for specific channel in a block (hasil -> MIUr MIUg MIUb)
7      public static double[] calculateAllMeans(BufferedImage image, int x, int y, int width, int height) {
8          double[] meanslist = new double[3];
9          long[] sums = {0, 0, 0};
10         int N = width * height;
11
12         for (int j = y; j < height + y; j++) {
13             for (int i = x; i < width + x; i++) {
14                 int rgb = image.getRGB(i, j);
15                 Color color = new Color(rgb);
16
17                 sums[redChannel] += color.getRed();
18                 sums[greenChannel] += color.getGreen();
19                 sums[blueChannel] += color.getBlue();
20             }
21         }
22         for (int i = 0; i < 3; i++) {
23             meanslist[i] = (double) sums[i] / N;
24         }
25         return meanslist;
26     }
```

Gambar 4.2.1 Kelas ChannelUtil (*calculateAllMeans*)

```

1  // * Get pixel intensity
2  public static List<Integer> getChannelValues(BufferedImage image, int x, int y, int width, int height, int channel) {
3      List<Integer> values = new ArrayList<>(width * height);
4
5      for (int j = y; j < height + y; j++) {
6          for (int i = x; i < width + x; i++) {
7              int rgb = image.getRGB(i, j);
8              Color color = new Color(rgb);
9
10             switch (channel) {
11                 case redChannel:
12                     values.add(color.getRed());
13                     break;
14                 case greenChannel:
15                     values.add(color.getGreen());
16                     break;
17                 case blueChannel:
18                     values.add(color.getBlue());
19                     break;
20             }
21         }
22     }
23     return values;
24 }
25
26 // * Get min max for MAD error calculation
27 public static double[] getChannelMinMax(BufferedImage image, int x, int y, int width, int height, int channel) {
28     int min = 255;
29     int max = 0;
30
31     for (int j = y; j < height + y; j++) {
32         for (int i = x; i < width + x; i++) {
33             int rgb = image.getRGB(i, j);
34             Color color = new Color(rgb);
35
36             int value = 0;
37             switch (channel) {
38                 case redChannel:
39                     value = color.getRed();
40                     break;
41                 case greenChannel:
42                     value = color.getGreen();
43                     break;
44                 case blueChannel:
45                     value = color.getBlue();
46                     break;
47             }
48
49             if (value < min) {
50                 min = value;
51             }
52
53             if (value > max) {
54                 max = value;
55             }
56         }
57     }
58     return new double[] {min, max};
59 }
60 }

```

Gambar 4.2.2 Kelas ChannelUtil (metode *getChannelValues* dan *getChannelMinmax*)

Selanjutnya, setiap perhitungan sesuai metode error yang dijelaskan pada Bab III Teori Singkat sebelumnya, diimplementasikan pada kelas **ErrorCalculation.java**. Setiap metode di bawah mengimplementasikan formula perhitungan error sesuai dengan rumus pada spesifikasi tugas, serta menerima parameter seperti *list of integer values* (berisi list intensitas channel RGB), beserta mean, dan minMax yang semua dikalkulasi mengutilisasi kelas ChannelUtil sebelumnya. Metode-metode perhitungan tersebut dapat dilihat di gambar di bawah ini.

```

1  class ErrorCalculation {
2      private static final double LOG2 = 1.442695;
3      // * Error calculation methods (per channel)
4      // Variance per channel
5      private static double calculateVarianceForChannel(List<Integer> values, double mean) {
6          if (values.size() == 0) {return 0.0;}
7          double squaredDiff = 0;
8
9          for (int value: values) {
10             double diff = value - mean;
11             squaredDiff += diff * diff;
12         }
13         return squaredDiff / values.size();
14     }
15
16     // MAD per channel
17     public static double calculateMADForChannel(List<Integer> values, double mean) {
18         if (values.size() == 0) {return 0.0;}
19         double absDiff = 0;
20         for (int value: values) {
21             absDiff += Math.abs(value - mean);
22         }
23         return absDiff / values.size();
24     }
25
26     // Max Pixel Difference per channel
27     public static double calculateMPDForChannel(double[] minMax) {
28         if (minMax.length == 0) {return 0.0;}
29         if (minMax.length == 1) {return 0.0;}
30
31         return minMax[1] - minMax[0];
32     }
33
34     // Entropi per channel
35     private static double calculateEntropyForChannel(List<Integer> values) {
36         int[] freq = new int[256];
37         if (values.size() == 0) {return 0.0;}
38         for (int value : values) {
39             freq[value]++;
40         }
41
42         double entropy = 0.0;
43         for (int count : freq) {
44             if (count > 0) {
45                 double p = (double) count / values.size();
46                 entropy += p * (Math.log(p) * LOG2); // log2(p) = ln(p)/ln(2)
47             }
48         }
49         return -entropy;
50     }

```

Gambar 4.2.3 Metode perhitungan per-channel setiap metode error dalam kelas

ErrorCalculation

Selanjutnya, hasil error per channel dan rata-rata tiap piksel (*mean*) dipakai untuk disimpan bagi setiap channel RGB, dan error tersebut akan didapatkan nilai rata-ratanya (nilai error R, G, B dijumlah dan dibagi 3) dalam metode **getError()**. Metode ini menerima node Quadtree, image, beserta pilihan metode error (*errorMethod*) dari user, dengan tujuan menyediakan fungsi perhitungan error method yang nanti diaplikasikan secara rekursif untuk setiap node Quadtree dan dibandingkan dengan *threshold* nantinya untuk syarat splitting.


```

1 public static double getError(QuadTreeNode node, BufferedImage image, int errorMethod) {
2     double error = 0;
3     int height = node.getHeight();
4     int width = node.getWidth();
5     int x = node.getX();
6     int y = node.getY();
7
8     if (height == 0 || width == 0) {
9         return 0;
10    }
11
12    List<Integer> redValues = ChannelUtil.getChannelValues(image, x, y, width, height, ChannelUtil.redChannel);
13    List<Integer> greenValues = ChannelUtil.getChannelValues(image, x, y, width, height, ChannelUtil.greenChannel);
14    List<Integer> blueValues = ChannelUtil.getChannelValues(image, x, y, width, height, ChannelUtil.blueChannel);
15
16    switch (errorMethod) {
17        case 1:
18            // Variance
19            double[] meansList = ChannelUtil.calculateAllMeans(image, x, y, width, height);
20            double meanR = meansList[0];
21            double meanG = meansList[1];
22            double meanB = meansList[2];
23            double varR = calculateVarianceForChannel(redValues, meanR);
24            double varG = calculateVarianceForChannel(greenValues, meanG);
25            double varB = calculateVarianceForChannel(blueValues, meanB);
26
27            error = (varR + varG + varB) / 3.0;
28            break;
29
30        case 2:
31            // MAD
32            meansList = ChannelUtil.calculateAllMeans(image, x, y, width, height);
33            meanR = meansList[0];
34            meanG = meansList[1];
35            meanB = meansList[2];
36
37            double madR = calculateMADForChannel(redValues, meanR);
38            double madG = calculateMADForChannel(greenValues, meanG);
39            double madB = calculateMADForChannel(blueValues, meanB);
40
41            error = (madR + madG + madB) / 3.0;
42            break;
43    }
44 }

```

Gambar 4.2.4 Metode getError dan perhitungan rata-rata untuk metode Variance dan Mean Absolute Deviation (MAD)

```

1
2     case 3:
3         // Max Pixel Difference
4         double[] minMaxR = ChannelUtil.getChannelMinMax(image, x, y, width, height, ChannelUtil.redChannel);
5         double[] minMaxG = ChannelUtil.getChannelMinMax(image, x, y, width, height, ChannelUtil.greenChannel);
6         double[] minMaxB = ChannelUtil.getChannelMinMax(image, x, y, width, height, ChannelUtil.blueChannel);
7
8         double mpdR = calculateMPDForChannel(minMaxR);
9         double mpdG = calculateMPDForChannel(minMaxG);
10        double mpdB = calculateMPDForChannel(minMaxB);
11        error = (mpdR + mpdG + mpdB) / 3.0;
12        break;
13
14    case 4:
15        // Entropy
16        double entropyR = calculateEntropyForChannel(redValues);
17        double entropyG = calculateEntropyForChannel(greenValues);
18        double entropyB = calculateEntropyForChannel(blueValues);
19        error = (entropyR + entropyG + entropyB) / 3.0;
20        break;
21
22    case 5:
23        // SSIM (Simplified, luminance only)
24        meansList = ChannelUtil.calculateAllMeans(image, x, y, width, height);
25        meanR = meansList[0];
26        meanG = meansList[1];
27        meanB = meansList[2];
28        varR = calculateVarianceForChannel(redValues, meanR);
29        varG = calculateVarianceForChannel(greenValues, meanG);
30        varB = calculateVarianceForChannel(blueValues, meanB);
31
32        final int L = 255;
33        final double K2 = 0.03;
34        final double wR = 0.299;
35        final double wG = 0.587;
36        final double wB = 0.114;
37
38        double C2 = Math.pow(K2 * L, 2);
39
40        double ssimR = C2 / (varR + C2);
41        double ssimG = C2 / (varG + C2);
42        double ssimB = C2 / (varB + C2);
43        double ssimTotal = wR * ssimR + wG * ssimG + wB * ssimB;
44        error = 1 - ssimTotal;
45        break;
46    }
47
48    return error;

```

Gambar 4.2.5 Metode `getError` dan perhitungan rata-rata untuk metode Max Pixel Difference, Entropy, dan Structural Similarity Index Measure (SSIM)

Di gambar 4.2.5 juga diimplementasikan metode perhitungan SSIM yang akan dijelaskan lebih detail di Bab V Implementasi Bonus. Dengan menghitung error untuk metode error pilihan, nilai ini kemudian akan dipakai dalam kelas **Compressor**, di mana metode algoritma *divide and conquer* utama untuk kompresi gambar akan diimplementasikan.

4.3 Proses kompresi gambar dengan Algoritma Divide and Conquer

Proses kompresi utama dengan algoritma *divide and conquer* terdapat pada kelas **Compressor**, di mana kelas ini menyimpan atribut-atribut utama seperti *imageInfo* (Buffered Image), *root* (akar Quadtree atau node pertama), *nodeCnt* (jumlah simpul total termasuk root), *tolerance* (toleransi perbedaan size untuk fitur target percentage), dan *executionTime* (lama eksekusi). Setelah itu, kelas Compressor juga menyimpan kebutuhan lain untuk fitur bonus GIF.

Kelas ini mengkonstruksi awal dengan *imageInfo* sebagai image original yang diinput oleh user. Lalu, metode **compress()** menjadi fungsi utama dalam menjalankan proses, diawali dengan 2 kondisi yaitu ketika fitur target percentage dalam mode on dan tidak (ini akan dijelaskan lebih detail pada bab selanjutnya). Metode ini bekerja secara rekursif untuk membagi node Quadtree menjadi 4 hingga batasan tertentu dengan:

1. Menginisialisasikan root node dengan $x=0$, $y=0$, dan height/width, sesuai dengan original image serta *maxDepth* (kedalaman tree) dan *nodeCnt*(jumlah node) sama dengan 1, node ini akan dipecah jika tidak memenuhi kriteria keseragaman.
2. Setelah itu, akan dibuild tree baru secara rekursif untuk setiap node dengan metode **buildTree()** (*divide and conquer* utama) yang menerima node Quadtree dan kedalaman Quadtree sebagai parameter. Metode *buildTree* akan memanggil metode **shouldSplit()**, yaitu metode yang menentukan apakah pohon perlu dibagi lebih lanjut, dan jika tidak maka node akan dideklarasikan sebagai leaf node (tidak perlu dibagi lagi). Setiap membagi, maka kedalaman (*maxDepth*) akan ditambahkan 1
3. Metode **shouldSplit** memiliki syarat, yaitu node akan membagi jika *error* > *threshold* dan ukuran node setelah dibagi 4 lebih besar dari minimum block size (*minBlockSize*). Pada implementasi di program ini, tidak diberikan kondisi akan membagi ketika ukuran *node* > *minBlockSize* karena jika ukuran node setelah dibagi 4 lebih besar dibanding *minBlockSize*, pastinya ukuran node itu sendiri sebelum dibagi juga lebih besar. Jika semua syarat ini terpenuhi, maka node akan dipecah

menjadi 4 dalam metode `buildTree` sebelumnya. Metode ini juga menghentikan secara awal jika image tidak bisa dibagi lagi.

4. Selain membuat node sebagai leaf ketika suatu blok tidak perlu dibagi lagi, metode `buildTree` menetapkan average node (`setNodeColor()`, metode terdapat pada `QuadTreeNode`) untuk menyimpan warna rata-rata dari blok di node.
5. Setelah itu, metode `createCompressedImage()` digunakan untuk membuat image terkompresi dan akan diwarnakan melalui metode `colorImage()`. Metode `colorImage` menerima parameter node dan target (*BufferedImage*) dan akan secara rekursif berulang mewarnakan node daun (*leaf*) dengan warna rata-rata dari setiap node (yang sebelumnya di set oleh `setNodeColor`. Proses ini lanjut hingga algoritma sudah membagi node/blok hingga tidak bisa dibagi lagi.
6. Lama eksekusi (*executionTime*) akan didapatkan dan ditetapkan kembali beserta dengan atribut lainnya seperti kedalaman Quadtree (*maxDepth*) dan jumlah total simpul (*nodeCnt*).

```
1 public class Compressor {
2     private ImageInfo imageInfo;
3     private QuadTreeNode root;
4     private int nodeCnt;
5     private int maxDepth;
6     private long executionTime;
7     private String gifPath;
8     private GifWriter gifWriter;
9
10    private final int tolerance = 100;
11
12    // * Ctor
13    public Compressor(ImageInfo imageInfo) {
14        this.imageInfo = imageInfo;
15        this.gifPath = imageInfo.getGifPath();
16    }
17
18    // * Methods
19    public BufferedImage createCompressedImage() {
20        if (root == null) {
21            throw new IllegalStateException("Error: Compression tree is not built yet.");
22        }
23        BufferedImage compressedImage = new BufferedImage(imageInfo.getOriginalImage().getWidth(),
24            imageInfo.getOriginalImage().getHeight(), imageInfo.getOriginalImage().getType());
25        colorImage(root, compressedImage);
26        return compressedImage;
27    }
28
29    // * Compress
30    public void compress() {
31        if (imageInfo.isTargetPercentageNode()) {
32            compressToTargetPercentage();
33            return;
34        }
35        // System.out.println("testafter");
36        long startTime = System.nanoTime();
37
38        BufferedImage image = imageInfo.getOriginalImage();
39        this.root = new QuadTreeNode(0, 0, image.getWidth(), image.getHeight());
40        this.maxDepth = 1;
41        this.nodeCnt = 1;
42
43
44        buildTree(root, 1);
45
46        long endTime = System.nanoTime();
47        this.executionTime = (endTime - startTime) / 1000000;
48    }
49 }
```

Gambar 4.3.1 Kelas `Compressor` dengan konstruktor, metode `createCompressedImage`, dan metode `compress`

```

1 // Syarat split check
2 private boolean shouldSplit(QuadTreeNode node) {
3     if (node.getWidth() < 2 || node.getHeight() < 2) {
4         return false;
5     } // stop kalau block sudah terlalu kecil
6
7     double error = ErrorCalculation.getError(
8         node,
9         imageInfo.getOriginalImage(),
10        imageInfo.getErrorMethod()
11    );
12
13    return error > imageInfo.getThreshold() &&
14        ((node.getHeight()/2) * (node.getWidth()/2) >= imageInfo.getMinBlockSize());
15 }
16
17 // DnC algorithm
18 private void buildTree(QuadTreeNode node, int currDepth) {
19
20     if (shouldSplit(node)) {
21         node.split();
22         nodeCnt += 4;
23         maxDepth = Math.max(maxDepth, currDepth + 1);
24
25         buildTree(node.getTopLeft(), currDepth + 1);
26         buildTree(node.getTopRight(), currDepth + 1);
27         buildTree(node.getBottomLeft(), currDepth + 1);
28         buildTree(node.getBottomRight(), currDepth + 1);
29     } else {
30         setNodeColor(node);
31         node.setIsLeaf(true);
32     }
33 }

```

Gambar 4.3.2 Metode shouldSplit untuk syarat pembagian node dan metode buildTree untuk membangun tree secara rekursif

```

1 // > HELPER FUNCTIONS
2 // Set avg rgb for each node
3 public void setNodeColor(QuadTreeNode node) {
4     BufferedImage image = imageInfo.getOriginalImage();
5     int height = node.getHeight();
6     int width = node.getWidth();
7     int x = node.getX();
8     int y = node.getY();
9
10    double[] meansList = ChannelUtil.calculateAllMeans(image, x, y, width, height);
11    node.setMeanValues(meansList[0], meansList[1], meansList[2]);
12 }
13
14 // fill the new image with color
15 public void colorImage(QuadTreeNode node, BufferedImage target) {
16     if (node == null) {
17         return;
18     }
19     if (node.getIsLeaf()) {
20         Color color = new Color(
21             Math.round((float)node.getMeanR()),
22             Math.round((float)node.getMeanG()),
23             Math.round((float)node.getMeanB())
24         );
25         for (int y = node.getY(); y < node.getY() + node.getHeight(); y++) {
26             for (int x = node.getX(); x < node.getX() + node.getWidth(); x++) {
27                 target.setRGB(x, y, color.getRGB());
28             }
29         }
30     } else {
31         colorImage(node.getTopLeft(), target);
32         colorImage(node.getTopRight(), target);
33         colorImage(node.getBottomLeft(), target);
34         colorImage(node.getBottomRight(), target);
35     }
36 }

```

Gambar 4.3.3 Metode setNodeColor untuk menyimpan rata-rata intensitas warna dan metode colorImage untuk mewarnai Image dengan rata-rata warna setiap node

4.4 Penyelesaian dan Pengolahan Hasil

Dengan demikian, setelah sudah rekursi sampai semua blok tidak bisa dibagi, maka hasil kompresi (berserta dengan GIF file) akan di save ke alamat direktori sesuai dengan input user. Selain itu, statistik kompresi seperti waktu eksekusi, ukuran file sebelum dan sesudah kompresi, kedalaman pohon Quadtree, serta jumlah total simpul juga disimpan dan ditampilkan kepada pengguna untuk memberikan gambaran performa algoritma. Seluruh proses ini memastikan bahwa hasil kompresi tidak hanya optimal secara ukuran, namun juga terkontrol kualitasnya berdasarkan parameter yang telah diinputkan. Efisiensi program ini dikembangkan lebih lanjut melalui fitur-fitur tambahan yang akan dijelaskan di dalam bab selanjutnya.

```
-----  
|                                COMPRESSION RESULTS                                |  
| IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree             |  
-----  
Compressed image saved successfully at: [a.png]!  
  
<< Execution time: 3604 ms >>  
  
Original file size: 46605 bytes  
Compressed file size: 32624 bytes  
Compression percentage: 29.999%  
Quadtree depth: 11  
Node count: 10641
```

Gambar 4.4.1 Contoh tampilan akhir menunjukan hasil kompresi dan statistik kompresi lainnya

4.5 Kompleksitas Algoritma

Fungsi utama `buildTree()` berjalan secara rekursif. Setiap blok gambar akan dibagi menjadi empat bagian jika tidak memenuhi kriteria keseragaman warna. Dalam kasus terburuk, seluruh gambar akan dibagi hingga ke blok-blok terkecil. Setiap pemrosesan node membutuhkan waktu linear terhadap jumlah piksel dalam bloknya, sehingga kompleksitas waktunya secara keseluruhan:

$$O(N^2 \cdot \log N)$$

Struktur Quadtree menyimpan node yang mewakili blok-blok gambar. Dalam kasus paling ekstremnya (semua blok dibagi hingga ukuran terkecil baru stop), jumlah node maksimal sekitar $\frac{N^2}{minBlockSize^2}$ sehingga untuk kompleksitas ruang, kode ini membutuhkan kompleksitas:

$$O(N^2)$$

BAB V

IMPLEMENTASI BONUS

Selain spesifikasi wajib, dalam proyek ini tiga fitur bonus diimplementasikan, yaitu persentase kompresi atau *Target Compression Percentage*, metode pengukuran error Structural Similarity Index (SSIM), dan output berupa GIF visualisasi proses pembentukan Quadtree dalam kompresi gambar.

5.1 Target Persentase Kompresi

Fitur target persentase kompresi sebagai input memungkinkan pengguna untuk menentukan target ukuran image hasil kompresi. Untuk menggunakan mode ini, perlu sebuah input berupa nilai *floating point* dengan range 0 sampai 1 yang menandakan target kompresi 0% hingga 100% (misalnya $0.7 = 70\%$). Jika diinput nilai 0, maka mode ini akan dinonaktifkan. Jika tidak 0, maka program masuk ke mode khusus `compressToTargetPercentage()`. Tujuan dari fitur ini adalah agar user dapat mengatur kualitas visual maupun ukuran file secara otomatis tanpa harus menentukan nilai threshold secara manual.

Cara kerja fitur ini adalah program akan secara otomatis mencari nilai threshold optimal menggunakan *binary search*, salah satu implementasi algoritma *divide and conquer*. Perubahan threshold akan mengubah file size hasil kompresi. Threshold yang terlalu kecil akan menghasilkan file size yang besar ukurannya, sedangkan threshold yang terlalu besar akan menghasilkan ukuran gambar yang terlalu kecil dan tidak sesuai target. Oleh karena itu, dengan metode ini, dilakukan penyesuaian dalam rentang minimum-maksimum sesuai metode error tertentu, karena karakteristik masing-masing metode memang menghasilkan skala error yang berbeda. Ukuran target (*targetSize*) ditentukan dari rumus yang diturunkan dari perhitungan pada subbab 3.4, yaitu $targetSize = ((1 - targetRatio) * originalSize)$. dan pencarian threshold dihentikan jika selisih antara ukuran kompresi dan target kurang dari toleransi 100 byte, atau saat iterasi mencapai batas maksimum (30 kali).

Binary search dapat digunakan secara efisien dalam program ini untuk mempersempit ruang pencarian threshold yang optimal, karena hubungan antara nilai threshold dan ukuran gambar bersifat monotonik (semakin besar threshold, maka kompresi lebih tinggi, dan ukuran lebih kecil). Algoritma *binary search* akan mencari threshold terbaik agar ukuran file mendekati target (pada program ini, mendekati toleransi yang ditentukan yaitu 100 bytes).

Setiap metode pengukuran error memiliki metode error yang berbeda, sehingga karakteristik yang juga berbeda. Oleh karena itu, dalam program ini diatur rentang threshold (minimum dan maksimum) yang spesifik untuk masing-masing metode. Setelah itu, program melakukan uji coba terhadap nilai minThreshold dan maxThreshold untuk memastikan bahwa ukuran target berada dalam jangkauan keduanya. Jika ukuran target memungkinkan, program melakukan proses binary search iteratif terhadap nilai threshold terbaik dan membuild ulang tree jika belum mencapai target. Dalam program, dibuat juga maxIteration sebanyak 30 kali sebagai batas iterasi jika program masih belum berhasil menyesuaikan untuk mencegah *infinite loop*.

```

1  // * Target Percentage Feature (BONUS)
2  public void compressToTargetPercentage() {
3      long startTime = System.nanoTime();
4
5      BufferedImage image = imageInfo.getOriginalImage();
6      long originalSize = imageInfo.getInputSize(imageInfo.getInputPath());
7
8      double targetRatio = imageInfo.getTargetCompressionPercentage();
9      long targetSize = (long) ((1 - targetRatio) * originalSize);
10
11      System.out.println("Original size: " + originalSize + " bytes, Target size: " + targetSize + " bytes");
12      System.out.println("\nPlease wait, this may take a while...");
13
14      // initial threshold (based on effective range in error method divided by 2)
15      double threshold;
16      double minThreshold, maxThreshold;
17
18      switch (imageInfo.getErrorMethod()) {
19          case 1: // var
20              threshold = 8192;
21              minThreshold = 1;
22              maxThreshold = 100000;
23              break;
24          case 2: // mad
25              threshold = 64;
26              minThreshold = 1;
27              maxThreshold = 1000;
28              break;
29          case 3: // mpd
30              threshold = 128;
31              minThreshold = 1;
32              maxThreshold = 1000;
33              break;
34          case 4: // entropy
35              threshold = 4;
36              minThreshold = 0.1;
37              maxThreshold = 50;
38              break;
39          case 5: // ssim
40              threshold = 0.5;
41              minThreshold = 0.01;
42              maxThreshold = 1.0;
43              break;
44          default:
45              threshold = 8192;
46              minThreshold = 1;
47              maxThreshold = 100000;
48              break;
49      }
50
51      long minSize = 0;
52      long maxSize = 0;

```

Gambar 5.1.1 Metode compressToTargetPercentage dan perhitungan targetSize, serta rentang threshold untuk metode perhitungan error yang berbeda

```

1  try {
2      // Test minimum threshold & juga maximum threshold
3      imageInfo.setThreshold(minThreshold);
4      this.root = new QuadTreeNode(0, 0, image.getWidth(), image.getHeight());
5      this.maxDepth = 1;
6      this.nodeCnt = 1;
7      buildTree(root, 1);
8      BufferedImage minImage = createCompressedImage();
9      minSize = getImageSize(minImage, imageInfo.getInputFormat());
10
11     imageInfo.setThreshold(maxThreshold);
12     this.root = new QuadTreeNode(0, 0, image.getWidth(), image.getHeight());
13     this.maxDepth = 1;
14     this.nodeCnt = 1;
15     buildTree(root, 1);
16     BufferedImage maxImage = createCompressedImage();
17     maxSize = getImageSize(maxImage, imageInfo.getInputFormat());
18
19     boolean isTargetAchievable = (minSize >= targetSize && maxSize <= targetSize) ||
20                                   (minSize <= targetSize && maxSize >= targetSize);
21
22     if (!isTargetAchievable) {
23         // use closest threshold
24         if (Math.abs(minSize - targetSize) < Math.abs(maxSize - targetSize)) {
25             threshold = minThreshold;
26         } else {
27             threshold = maxThreshold;
28         }
29     } else {
30         int maxIter = 30;
31         int iter = 0;
32         double currThreshold = threshold;
33
34         while (iter < maxIter) {
35             imageInfo.setThreshold(currThreshold);
36
37             // Reset tree
38             this.root = new QuadTreeNode(0, 0, image.getWidth(), image.getHeight());
39             this.maxDepth = 1;
40             this.nodeCnt = 1;
41             buildTree(root, 1);
42
43             BufferedImage compressedImage = createCompressedImage();
44             long compressedSize = getImageSize(compressedImage, imageInfo.getInputFormat());
45
46             if (Math.abs(compressedSize - targetSize) <= tolerance) {
47                 threshold = currThreshold;
48                 break;
49             }
50
51             // binary search for best threshold
52             // ? size too large -> less compression -> more threshold
53             // ? size too small -> more compression -> less threshold
54             if (compressedSize > targetSize) {
55                 minThreshold = currThreshold;
56                 minSize = compressedSize;
57             } else {
58                 maxThreshold = currThreshold;
59                 maxSize = compressedSize;
60             }
61             currThreshold = (minThreshold + maxThreshold) / 2;
62             iter++;
63         }
64
65         threshold = currThreshold;
66     }

```

Gambar 5.1.2 Pencarian threshold optimal secara rekursif menggunakan binary search


```

1 // Final compress (with best threshold)
2     imageInfo.setThreshold(threshold);
3     this.root = new QuadTreeNode(0, 0, image.getWidth(), image.getHeight());
4     this.maxDepth = 1;
5     this.nodeCnt = 1;
6     buildTree(root, 1);
7
8     System.out.println("Final threshold: " + threshold);
9
10    long endTime = System.nanoTime();
11    this.executionTime = (endTime - startTime) / 1000000;
12 } catch (IOException e) {
13     System.err.println("Error: " + e.getMessage());
14 }
15 }

```

Gambar 5.1.3 Build tree terakhir dengan threshold yang optimal setelah keluar dari loop binary search

5.2 Structural Similarity Index Measure (SSIM)

Metode pengukuran error dalam bonus ini merupakan pengukuran error yang mempertimbangkan kualitas visual secara struktural. SSIM umumnya digunakan untuk menilai kemiripan antara dua gambar berdasarkan tiga komponen utama: luminance (penerangan), contrast (kontras), dan structure (struktur spasial). Nilai SSIM berada pada rentang 0 hingga 1, di mana nilai lebih tinggi menunjukkan kemiripan visual yang lebih baik.

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

Dalam program ini, rumus SSIM disederhanakan agar lebih efisien. Kompresi blok Quadtree yang dikompresi bernilai monoton, maka asumsi berikut dapat digunakan:

- $\mu_x = \mu_y$: rata-rata piksel blok original dan compressed dianggap sama
- $\sigma_y = 0$: karena blok hasil kompresi berisi warna seragam (monoton), maka tidak ada keragaman piksel.
- $\sigma_{xy} = 0$: kovariansi antar blok 0 karena salah satunya monoton

Dengan asumsi tersebut, rumus SSIM dapat disederhanakan menjadi:

$$SSIM = \frac{C_2}{\sigma_x^2 + C_2}$$

Konstanta C_2 dihitung melalui rumus:

$$C_2 = (K_2 \cdot L)^2$$

Di mana:

- $K_2 = 0,03$ (nilai konstanta konvensional SSIM)
- $L = 255$ (nilai maksimum piksel untuk gambar 8-bit sesuai spesifikasi)

Nilai SSIM berada dalam rentang 0 hingga 1 dengan 0 berarti blok sangat berbeda dan 1 berarti blok sangat mirip. Oleh karena itu, nilai SSIM yang tinggi berarti kualitas kompresi baik. Namun, dalam konteks Quadtree, blok dibagi berdasarkan besar kecilnya nilai error. Oleh karena itu, agar logika tetap konsisten seperti metode error lain, maka nilai error untuk SSIM didefinisikan sebagai $1 - SSIM$. Semakin kecil SSIM, semakin besar nilai error dan semakin besar kemungkinan blok akan dibagi lebih lanjut.

```
1  case 5:
2      // SSIM (Simplified, Luminance only)
3      meansList = ChannelUtil.calculateAllMeans(image, x, y, width, height);
4      meanR = meansList[0];
5      meanG = meansList[1];
6      meanB = meansList[2];
7      varR = calculateVarianceForChannel(redValues, meanR);
8      varG = calculateVarianceForChannel(greenValues, meanG);
9      varB = calculateVarianceForChannel(blueValues, meanB);
10
11     final int L = 255;
12     final double K2 = 0.03;
13     final double wR = 0.299;
14     final double wG = 0.587;
15     final double wB = 0.114;
16
17     double C2 = Math.pow(K2 * L, 2);
18
19     double ssimR = C2 / (varR + C2);
20     double ssimG = C2 / (varG + C2);
21     double ssimB = C2 / (varB + C2);
22     double ssimTotal = wR * ssimR + wG * ssimG + wB * ssimB;
23     error = 1 - ssimTotal;
24     break;
```

5.3 Output berupa GIF Visualisasi Proses

Fitur output berupa GIF, dilakukan dengan fungsi **createGif()** pada kelas `Compressor.java`, dibantu dengan kelas **GifWriter** yang bertugas menangani proses penulisan-frame-frame ke dalam file GIF. Kedua kelas ini memanfaatkan `ImageIO` untuk menulis urutan gambar sebagai animasi dan menyisipkan metadata seperti delay antar frame serta pengaturan loop agar GIF dapat diputar ulang secara terus-menerus.



Cara kerja fitur ini adalah, animasi GIF akan menampilkan tahapan-tahapan kompresi gambar dari kedalaman (*depth*) Quadtree tersebut. Proses mulai dari kedalaman tree paling atas hingga kedalaman maksimal. Setiap frame GIF merepresentasikan hasil kompresi pada stage kedalaman tertentu. Pada setiap iterasi depth, fungsi `createFrameEachDepth()` menggambarkan seluruh node yang telah menjadi node daun (*leaf*) dengan warna rata-rata dari blok tersebut. Setelah itu pada kelas `GifWriter`, setiap frame ditambahkan ke dalam urutan GIF menggunakan fungsi **writeToSequence()** dan ditutup dengan **endWriteSequence()** untuk menyelesaikan GIF. Frame terakhir dari GIF akan menampilkan hasil akhir kompresi gambar.

BAB VI



PENGETESAN PROGRAM

(Link seluruh GIF terlampir di folder /test dalam repository)

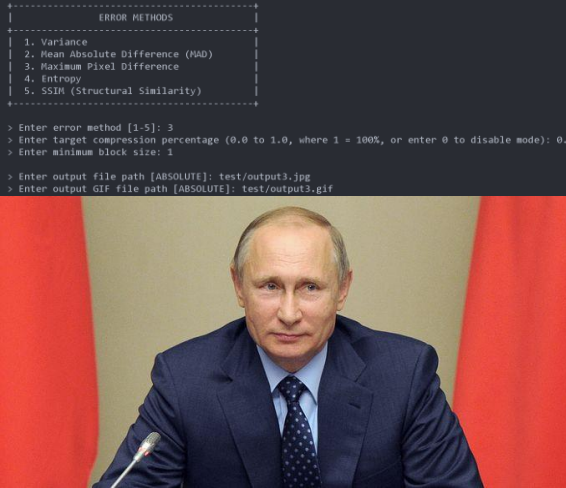
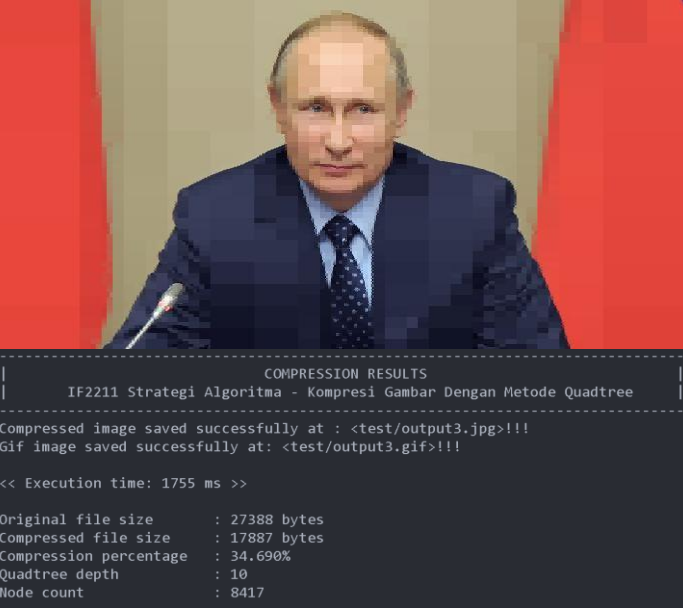
Test Case 1

Input	Output
<p>input1.png Method: Variance TargetCompression: ON (0.5 / 50%) Threshold: - Minimum Block Size: 1</p> <pre> ----- ERROR METHODS ----- 1. Variance 2. Mean Absolute Difference (MAD) 3. Maximum Pixel Difference 4. Entropy 5. SSIM (Structural Similarity) ----- > Enter error method [1-5]: 1 > Enter target compression percentage (0.0 to 1.0, where 1 = 100%, or enter 0 to disable mode): 0.5 > Enter minimum block size: 1 > Enter output file path [ABSOLUTE]: test/output1.png > Enter output GIF file path [ABSOLUTE]: test/output1.gif </pre> 	<p>output1.png</p>  <pre> ----- COMPRESSION RESULTS ----- IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree ----- Compressed image saved successfully at : <test/output1.png>!!! Gif image saved successfully at: <test/output1.gif>!!! << Execution time: 5921 ms >> Original file size : 355943 bytes Compressed file size : 178029 bytes Compression percentage : 49.984% Quadtree depth : 11 Node count : 70001 </pre>



Test Case 2

Input	Output
<p>input2.jpg Method: MAD TargetCompression: OFF Threshold: 50 Minimum Block Size: 4</p> <pre> +-----+ COMPRESSION SETTINGS +-----+ Selected Method: [2] +-----+ IDEAL THRESHOLD RANGES: [1] Variance: 0 - 16834.0 [2] MAD: 0 - 127.5 [3] MaxPixelDifference: 0 - 255.0 [4] Entropy: 0 - 8.0 [5] SSIM: 0 - 1.0 +-----+ > Enter threshold: 20 > Enter minimum block size: 1 > Enter output file path [ABSOLUTE]: test/output2.jpg > Enter output GIF file path [ABSOLUTE]: test/output2.gif +-----+ </pre> 	<p>output2.jpg</p>  <pre> +-----+ COMPRESSION RESULTS +-----+ IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree +-----+ Compressed image saved successfully at : <test/output2.jpg>!!! Gif image saved successfully at: <test/output2.gif>!!! << Execution time: 363 ms >> Original file size : 46605 bytes Compressed file size : 26953 bytes Compression percentage : 42.167% Quadtree depth : 11 Node count : 5685 +-----+ </pre>



Test Case 3

Input	Output
<p>input3.jpg Method: Maximum Pixel Difference TargetCompression: ON (0.35 / 35%) Threshold: - Minimum Block Size: 1</p> 	<p>output3.jpg</p> 

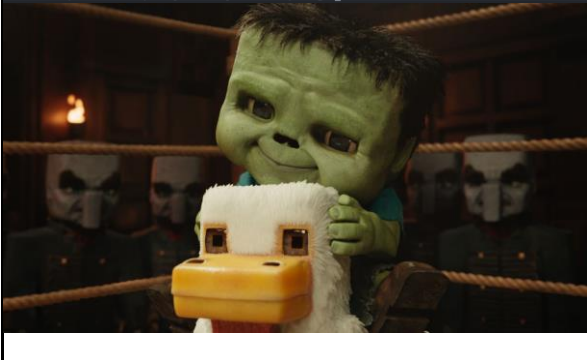
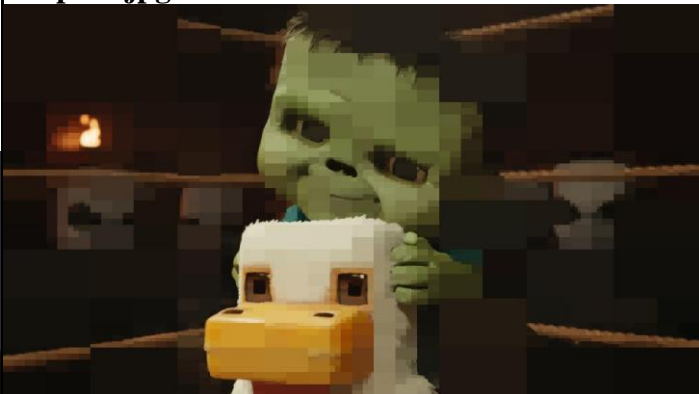
Test Case 4

Input	Output
<p>input4.png Method: Entropy TargetCompression: OFF Threshold: 2 Minimum Block Size: 4</p> <pre> +-----+ COMPRESSION SETTINGS +-----+ Selected Method: [4] +-----+ IDEAL THRESHOLD RANGES: [1] Variance: 0 - 16834.0 [2] MAD: 0 - 127.5 [3] MaxPixelDifference: 0 - 255.0 [4] Entropy: 0 - 8.0 [5] SSIM: 0 - 1.0 +-----+ > Enter threshold: 2 > Enter minimum block size: 4 > Enter output file path [ABSOLUTE]: test/output4.png > Enter output GIF file path [ABSOLUTE]: test/output4.gif </pre> 	<p>output4.png</p>  <pre> +-----+ COMPRESSION RESULTS +-----+ IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree +-----+ Compressed image saved successfully at : <test/output4.png>!!! Gif image saved successfully at: <test/output4.gif>!!! << Execution time: 589 ms >> Original file size : 557232 bytes Compressed file size : 150264 bytes Compression percentage : 73.034% Quadtree depth : 9 Node count : 45529 </pre>

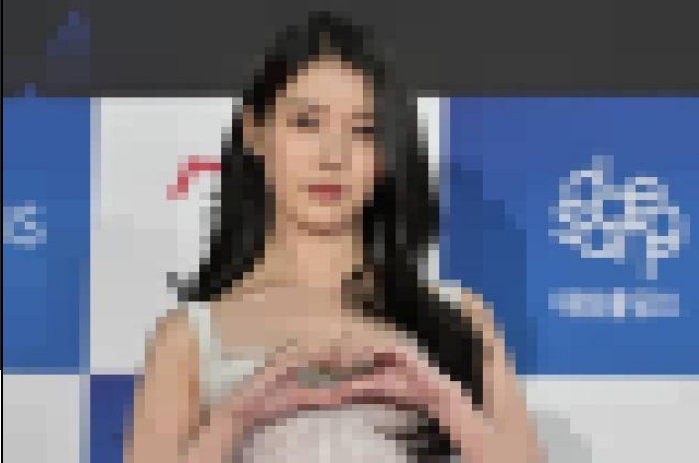
Test Case 5

Input	Output
<p>input5.jpg Method: SSIM TargetCompression: ON (0.43/43%) Threshold: - Minimum Block Size: 2</p> <pre> ===== ERROR METHODS ----- 1. Variance 2. Mean Absolute Difference (MAD) 3. Maximum Pixel Difference 4. Entropy 5. SSIM (Structural Similarity) ----- > Enter error method [1-5]: 5 > Enter target compression percentage (0.0 to 1.0, where 1 = 100%, or enter 0 to disable mode): 0.43 > Enter minimum block size: 2 > Enter output file path [ABSOLUTE]: test/output5.jpg > Enter output Gif file path [ABSOLUTE]: test/output5.gif </pre> 	<p>output5.jpg</p>  <pre> ===== COMPRESSION RESULTS ----- IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree ----- Compressed image saved successfully at : <test/output5.jpg>!!! Gif image saved successfully at: <test/output5.gif>!!! << Execution time: 4014 ms >> Original file size : 33906 bytes Compressed file size : 19469 bytes Compression percentage : 42.579% Quadtree depth : 9 Node count : 2645 </pre>

Test Case 6

Input	Output
<p>input6.jpg Method: Variance TargetCompression: ON (0.32/32%) Threshold: - Minimum Block Size: 1</p> <pre> ===== ERROR METHODS ----- 1. Variance 2. Mean Absolute Difference (MAD) 3. Maximum Pixel Difference 4. Entropy 5. SSIM (Structural Similarity) ----- > Enter error method [1-5]: 1 > Enter target compression percentage (0.0 to 1.0, where 1 = 100%, or enter 0 to disable mode): 0.32 > Enter minimum block size: 1 > Enter output file path [ABSOLUTE]: test/output6.jpg > Enter output GIF file path [ABSOLUTE]: test/output6.gif </pre> 	<p>output6.jpg</p>  <pre> ===== COMPRESSION RESULTS ----- IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree ----- Compressed image saved successfully at : <test/output6.jpg>!!! Gif image saved successfully at: <test/output6.gif>!!! << Execution time: 19796 ms >> Original file size : 89516 bytes Compressed file size : 61222 bytes Compression percentage : 31.608% Quadtree depth : 11 Node count : 13689 </pre>

Test Case 7

Input	Output
<p>input7.jpeg Method: Variance TargetCompression: OFF Threshold: 70 Minimum Block Size: 20</p> <pre> +-----+ COMPRESSION SETTINGS +-----+ Selected Method: [1] +-----+ IDEAL THRESHOLD RANGES: [1] Variance: 0 - 16834.0 [2] MAD: 0 - 127.5 [3] MaxPixelDifference: 0 - 255.0 [4] Entropy: 0 - 8.0 [5] SSIM: 0 - 1.0 +-----+ > Enter threshold: 70 > Enter minimum block size: 20 </pre>	<p>output7.jpeg</p> <pre> ----- COMPRESSION RESULTS IF2211 Strategi Algoritma - Kompresi Gambar Dengan Metode Quadtree ----- Compressed image saved successfully at : <c:\Users\andre\OneDrive\Documents\GitHub\Tuc112_13523148\test\output7.jpeg>!! Gif image saved successfully at: <c:\Users\andre\OneDrive\Documents\GitHub\Tuc112_13523148\test\output7.gif>!!! << Execution time: 257 ms >> Original file size : 40276 bytes Compressed file size : 24116 bytes Compression percentage : 40.123% Quadtree depth : 7 Node count : 2953 </pre> 

BAB VII

KESIMPULAN DAN SARAN

7.1 Kesimpulan

Implementasi penyelesaian Quadtree Image Compressor dilakukan dalam Tugas Kecil 2 IF2211 Strategi Algoritma dengan menggunakan algoritma *divide and conquer*. Program berhasil memenuhi spesifikasi wajib dengan memberikan solusi yang efektif dalam mengompresi gambar. Setelah itu, fitur bonus seperti target persentasi kompresi, output berupa GIF, dan error method SSIM dilakukan dengan selesai.

Dari hasil implementasi, dapat disimpulkan bahwa strategi *divide and conquer* melalui struktur *QuadTree* sangat efektif dalam mengatasi masalah kompresi gambar, terutama pada gambar dengan karakteristik pola yang berulang atau area homogen yang luas. Algoritma ini memungkinkan efisiensi dalam penyimpanan serta fleksibilitas dalam menyesuaikan tingkat kompresi dengan mengatur ambang batas. Selain itu, fitur visualisasi kompresi yang ditampilkan melalui animasi GIF memperkuat pemahaman pengguna terhadap proses rekursif yang terjadi selama pembentukan pohon.

7.2 Saran

Program dapat dikembangkan lebih lanjut, terutama dalam sisi optimalisasi kompresi yang dapat mempercepat proses kompresi dengan jauh lebih efisien, meningkatkan akurasi kompresi agar sesuai dengan kemauan user, ataupun menambahkan validasi input yang lebih lengkap dan konsisten untuk menghindari error.

BAB VIII

LAMPIRAN

Lampiran

Link Repository GitHub:

https://github.com/andrewtedja/Tucil2_13523148

Tabel Hasil

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	✓	