KOMPRESI GAMBAR DENGAN METODE QUADTREE

Laporan Tugas Kecil 2

Disusun untuk memenuhi mata kuliah IF2211 Strategi Algoritma

oleh:

Najwa Kahani Fatima 13523043

Nayla Zahira 13523079

PROGRAM STUDI TEKNIK INFORMATIKA

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA

INSTITUT TEKNOLOGI BANDUNG

BANDUNG

2025

DAFTAR ISI

BAB I

DESKRIPSI TUGAS DAN ALGORITMA

1.1. Quadtree dalam Kompresi Gambar

Quadtree merupakan struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian-bagian yang lebih kecil, yang umumnya diaplikasikan dalam berbagai bidang komputasi spasial, terutama dalam pengolahan citra digital. Dalam konteks kompresi gambar, Quadtree bekerja dengan membagi gambar menjadi blok-blok dengan ukuran bervariasi berdasarkan tingkat keseragaman warna atau intensitas piksel. Proses inisiasi dimulai dengan membagi gambar menjadi empat kuadran yang sama besar, kemudian algoritma melakukan evaluasi terhadap setiap kuadran untuk menentukan homogenitas area berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika suatu bagian tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

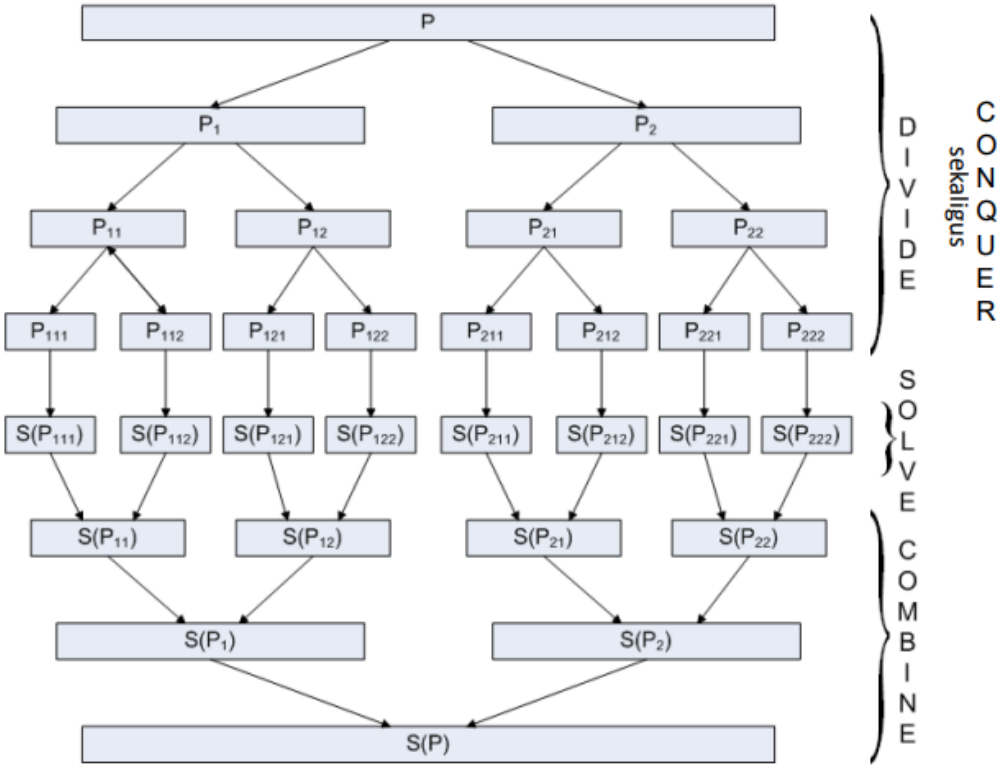
Secara teknis, sebuah Quadtree direpresentasikan sebagai struktur pohon di mana setiap simpul (*node*) dapat memiliki maksimal empat anak (*children*), yang masing-masing merepresentasikan area hasil pembagian. Simpul-simpul dalam Quadtree terbagi menjadi dua kategori utama: simpul daun (l*eaf node*) yang merepresentasikan area gambar yang seragam, dan simpul internal (*internal node*) yang mengindikasikan area yang masih memerlukan pembagian lebih lanjut. Setiap simpul dalam struktur Quadtree menyimpan metadata penting seperti koordinat posisi (x, y), dimensi area (*width, height*), serta nilai n nilai rata-rata warna atau intensitas piksel dalam area tersebut.

Implementasi Quadtree dalam kompresi gambar menawarkan beberapa keunggulan signifikan, antara lain kemampuan adaptasi terhadap kompleksitas lokal gambar, di mana area dengan detail tinggi akan mendapatkan resolusi yang lebih tinggi dibandingkan area yang homogen. Struktur ini telah terbukti efektif dalam berbagai aplikasi kompresi lossy karena kemampuannya dalam mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

1.2. Algoritma Divide and Conquer

Strategi algoritma *Divide and Conquer* merupakan pendekatan pemecahan masalah dengan cara memecah suatu persoalan besar menjadi beberapa sub-persoalan yang lebih kecil sehingga lebih mudah untuk diselesaikan. Strategi ini memiliki tiga tahapan yaitu *divide, conquer,* dan *combine*. Tahap pertama, proses *divide* membagi persoalan menjadi beberapa sub-persoalan yang memiliki kemiripan dengan persoalan semula tetapi berukuran lebih kecil. Kemudian, proses *conquer* berperan untuk menyelesaikan masing-masing sub-persoalan, baik secara langsung(jika sudah berukuran

kecil) maupun secara rekursif (jika masih berukuran besar). Terakhir, tahap *combine* berfungsi untuk menggabungkan solusi masing-masing sub-persoalan sehingga membentuk solusi persoalan semula.



Keterangan:
P = persoalan
S = solusi

1.3. Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree

Pada poin ini, penjelasan algoritma divide and conquer yang digunakan dalam program kompresi gambar akan dibagi menjadi 3 bagian sesuai dengan tahap-tahap algoritma divide and conquer yang telah dijelaskan pada poin sebelumnya.

1.3.1. Divide

Pada tahap ini, idenya adalah membagi blok menjadi 4 jika memenuhi 3 kondisi, yaitu hasil perhitungan perbedaan pada gambar di atas threshold, ukuran blok lebih besar dari ukuran blok minimum, dan ukuran blok setelah dibagi menjadi 4 tidak kurang dari ukuran blok minimum. Jika salah satu kondisi tidak terpenuhi, proses pembagian dihentikan. Berikut langkah-langkah dalam implementasinya.

- Hitung perbedaan pada blok yang sedang ditinjau menggunakan *error measurement method* yang telah dipilih sebelumnya.
- Hitung ukuran blok terkecil jika dibagi 4.

- Jika perbedaan pada blok masih diatas threshold dan ukuran blok terkecil saat dibagi 4 tidak kurang dari minimum block size, bagi blok menjadi 4 menggunakan batas tengah baris dan kolomnya
- Jika tidak memenuhi syarat untuk membagi blok, biarkan saja blok dalam kondisi saat ini

1.3.2. Conquer

Tahap ini dimulai ketika blok sudah tidak perlu dibagi lagi. Pada tahap ini, blok yang sudah tidak perlu dibagi lagi akan ditandai sebagai leaf node. Untuk setiap leaf node, nilai rata-rata dari komponen warna RGB dihitung menggunakan metode avgPixel(). Nilai rata-rata RGB ini kemudian disimpan dalam node sebagai representasi kompresi dari seluruh area blok. Proses ini merupakan inti dari kompresi gambar, di mana area dengan variasi warna yang rendah direpresentasikan oleh satu nilai warna.

1.3.3. Combine

Tahap combine terjadi ketika hasil dari sub-masalah digabungkan untuk membangun solusi akhir, yang dalam kasus ini adalah gambar terkompresi. Pada program ini, metode matrixFromQuadtree() berperan dalam tahap combine dengan membangun kembali gambar dari struktur Quadtree. Untuk setiap node leaf, nilai warna rata-rata RGB yang telah dihitung digunakan untuk mengisi seluruh piksel dalam blok yang sesuai pada gambar hasil. Sementara untuk node non-leaf, metode ini secara rekursif mengunjungi keempat child node (top-left, top-right, bottom-left, bottom-right) dan mengombinasikan hasil dari setiap kuadran untuk membentuk gambar utuh. Pendekatan ini menghasilkan gambar terkompresi di mana area dengan detail rendah direpresentasikan oleh blok warna seragam yang lebih besar, sedangkan area dengan detail tinggi direpresentasikan oleh blok-blok yang lebih kecil dan lebih terperinci.

1.4. Ilustrasi Pseudocode

```
function checker(integer[][] red, integer[][] green, integer[][] blue, integer rowTL,
integer colTL, integer h, integer w) -> Node
{ Jika memenuhi syarat  pembagian, matriks akan dibagi menjadi 4 node.  }

Deklarasi
   avgRed, avgGreen, avgBlue : integer
   node, nodeTL, nodeTR, nodeBL, nodeBR  : Node
   hHalf, wHalf, hRest, wRest : integer
   function isDivide(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int
   h, int w, int avgRed, int avgGreen, int avgBlue)
   {  menghasilkan true jika nilai perhitungan subblok dengan error measurement method
   yang dipilih melebihi threshold dan block area bagi 4 kurang dari minBlockSize  }

Algoritma:
    avgRed = avgPixel(red, rowTL, colTL, h, w);
```

```
    avgGreen = avgPixel(green, rowTL, colTL, h, w);
    avgBlue = avgPixel(blue, rowTL, colTL, h, w);
    node = new Node(avgRed, avgGreen, avgBlue, false, rowTL, colTL, h, w);
    this.numNode ++;

    if (!isDivide(red, green, blue, rowTL, colTL, h, w, avgRed, avgGreen, avgBlue)) then
        {  current blok sudah tidak bisa dibagi, current block menjadi leaf  }
        node.setIsLeaf(true);
        ->node;
    else
        hHalf ← h / 2;
        wHalf ← w / 2;
        hRest ← h - hHalf;
        wRest ← w - wHalf;

        {bagi menjadi 4 sub-blok}
        nodeTL ← checker(red, green, blue, rowTL, colTL, hHalf, wHalf);
        nodeTR ← checker(red, green, blue, rowTL, colTL + wHalf, hHalf, wRest);
        nodeBL ← checker(red, green, blue, rowTL + hHalf, colTL, hRest, wHalf);
        nodeBR ← checker(red, green, blue, rowTL + hHalf, colTL + wHalf, hRest, wRest);
        node.setTopLeft(nodeTL);
        node.setTopRight(nodeTR);
        node.setBotLeft(nodeBL);
        node.setBotRight(nodeBR);
        ->node;


procedure matrixFromQuadtree(Node node, integer[][] newImage, integer h, integer w)
{  membangun kembali matriks image dari node hasil divide and conquer  }
Deklarasi
   red, green, blue: integer
   node, nodeTL, nodeTR, nodeBL, nodeBR  : Node
   hHalf, wHalf, hRest, wRest : integer
   function isDivide(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int
   h, int w, int avgRed, int avgGreen, int avgBlue)
   {  menghasilkan true jika nilai perhitungan subblok dengan error measurement method
   yang dipilih melebihi threshold dan block area bagi 4 kurang dari minBlockSize  }

Algoritma:
    if (node.getIsLeaf()) then
        i traversal (node.getPos().row … node.getPos().row + h -1)
            j traversal (node.getPos().col … node.getPos().col + w -1)
                red ← node.getRedValue() << 16
                green ← node.getGreenValue() << 8
                blue ← node.getBlueValue()
                rgb ← red | green | blue
                newImage[i][j] ← rgb


    else
        hHalf ← h / 2
        wHalf ← w / 2
        hRest ← h - hHalf
        wRest ← w - wHalf

        matrixFromQuadtree(node.getTopLeft(), newImage, hHalf, wHalf)
        matrixFromQuadtree(node.getTopRight(), newImage, hHalf, wRest)
        matrixFromQuadtree(node.getBotLeft(), newImage, hRest, wHalf)
        matrixFromQuadtree(node.getBotRight(), newImage, hRest, wRest)
```

BAB II

IMPLEMENTASI PROGRAM

2.1. Struktur Kode Program

Pada program ini, sebagian kode yang krusial dibuat dengan paradigma berorientasi objek (OOP) untuk mempermudah pemrosesan metode atau fungsi yang merupakan tanggung jawab masing-masing kelas. Namun, terdapat beberapa kelas yang tidak menggunakan paradigma ini karena proses lebih bersifat prosedural. Folder program disusun sebagai berikut:

```
TUCIL2_13523043_13523079
├── bin/                          # Folder untuk file hasil kompilasi
├── doc/
│   └── laporan.pdf               # Laporan tugas dalam format PDF
├── src/                          # Folder source code utama
│   ├── errorMeasurement/         # Paket untuk pengukuran error
│   │   ├── Entropy.java
│   │   ├── MAD.java
│   │   ├── MPD.java
│   │   ├── SSIM.java
│   │   └── Variance.java
│   ├── tools/                    # Paket utilitas/tools
│   │   ├── GIF.java
│   │   ├── Image.java
│   │   ├── Input.java
│   │   ├── Node.java
│   │   ├── Point.java
│   │   ├── Quadtree.java
│   │   ├── TargettedCompressor.java
│   │   └── Tuple2.java
│   └── Main.java                 # Main class aplikasi
├── test/                         # Folder untuk hasil kompresi
│   ├── frames/                   # Folder penyimpanan frame GIF sementara
│   ├── compressed.jpg
│   └── compressed.gif
├── run.bat                       # Batch untuk compile dan run program
└── README.md                     # Dokumentasi utama proyek
```

2.2. Penjelasan Kelas-Kelas

2.2.1. Kelas Main

Kelas ini berisi program utama yang akan dijalankan untuk melakukan kompresi gambar menggunakan Quadtree.

2.2.2. Kelas GIF

Kelas ini berisi program untuk me-render GIF dari Quadtree. Dalam file yang sama, terdapat juga kelas GifSequence untuk melakukan konstruksi GIF dengan

parameter-parameter yang diberikan. Berikut adalah metode-metode yang ada pada kelas GIF dan GifSequence:

<table>
<tr><th colspan="2">Kelas GIF</th></tr>
<tr><th>Metode</th><th>Penjelasan</th></tr>
<tr><td><code>public static void makeGIF(Quadtree tree, String outputName)</code></td><td>Metode yang dipanggil di kelas Main untuk membuat GIF. Metode ini melakukan pemanggilan kode untuk membuat frame-frame sebagai konstruktor GIF dan membuat serta menyimpan GIF (.gif).</td></tr>
<tr><td><code>public static void getFrames(Quadtree tree)</code></td><td>Melakukan konstruksi frame (gambar) untuk setiap node pada kedalaman yang sama.</td></tr>
<tr><td><code>private static void getNodesBasedOnDepth(Node node, List<List<Node>> depthQueues, int depth)</code></td><td>Mengisi variabel depthQueues dengan List node-node pada kedalaman yang sama.</td></tr>
<tr><td><code>public static void updateMatrix(Node node, int[][] matrix)</code></td><td>Memperbarui matrix frame dengan atribut yang sesuai pada node (seperti letak nya pada matrix RGB, panjang kolom, dan panjang baris).</td></tr>
<tr><td><code>public static void saveFrame(int[][] matrix, int frameIndex, int h, int w)</code></td><td>Membuat frame baru (.png) ke folder test/frame/ untuk nanti dibuat menjadi GIF.</td></tr>
<tr><th colspan="2">Kelas GifSequence</th></tr>
<tr><td colspan="2">Atribut:

<code>private final ImageOutputStream output;
private final ImageWriter writer;
private final ImageTypeSpecifier typeSpecifier;
private final boolean isLoop; // true jika gif looping
private boolean isFirstFrame = true;</code></td></tr>
<tr><th>Metode</th><th>Penjelasan</th></tr>
<tr><td><code>public GifSequence(ImageOutputStream output, int imageType, boolean isLoop)</code></td><td>Konstruktor dari kelas GifSequence.</td></tr>
</table>

`private IIOMetadata getFrameMetadata(int delayMs)`	Metode untuk mengatur metadata setiap frame dalam GIF.
`public void writeToSequence(BufferedImage image, int delayMs)`	Menuliskan frame gambar ke GIF sequence.
`public void close()`	Menutup writer dan mengakhiri sequence GIF.

2.2.3. Kelas Image

Kelas ini digunakan untuk menyimpan dan memproses informasi dari gambar yang akan atau telah dikompresi.

Kelas Image	

Atribut:
 	
`private int width; // lebar gambar`
`private int height; // tinggi gambar`
`private int[][] red; // kanal merah`
`private int[][] green; // kanal hijau`
`private int[][] blue; // kanal biru`	

Metode	**Penjelasan**
`public Image(int width, int height, int[][] red, int[][] green, int[][] blue)`	Konstruktor kelas Image.
`public static Image loadImage(String imagePath)`	Metode untuk mengembalikan Image yang dimuat dari gambar pada *file path* tertentu. Metode ini juga mengekstrak kanal merah, biru, dan hijau pada gambar.
`public BufferedImage getCompressedImage(Quadtree quadtree)`	Metode untuk mengembalikan BufferedImage dari gambar terkompresi melalui Quadtree.
`public static int[][] extractCompressedMatrix(Quadtree quadtree)`	Metode untuk membuat dan memanggil metode untuk mengekstrak matriks RGB dari quadtree.
`public static void matrixFromQuadtree(Node`	Metode rekursif untuk

`node, int[][] newImage, int h, int w)`	mengekstrak nilai RGB dari masing-masing node.

`public int getWidth();`
`public int getHeight();`
`public int[][] getRed();`
`public int[][] getGreen();`
`public int[][] getBlue()`
	Getter dan setter atribut Image.

2.2.4. Kelas Input

Kelas ini digunakan untuk menerima dan memvalidasi input pengguna dalam kompresi gambar.

Kelas Input	
Metode	**Penjelasan**
`public static String readInputPath()`	Metode ini meminta pengguna memasukkan alamat absolut file gambar.
`public static int readInputErrorMethod (Scanner scanner)`	Metode ini menampilkan 5 metode perhitungan error yang tersedia dan menerima input pilihan metode berupa angka
`public static boolean isAbsolutePath(String path)`	Memeriksa apakah path yang diberikan adalah alamat absolut
`public static boolean isImageFile(String fileName)`	Memeriksa apakah file berupa file gambar
`public static double readInputThreshold(Scanner scanner)`	Menerima input threshold
`public static int readInputMinBlockSize(Scanner scanner, int imageBlockSize)`	Menerima input minimum block size
`public static double readInputTargetCompression(Scanner scanner)`	Menerima input target persentase kompresi
`public static String readOutputPath(Scanner scanner)`	Menerima input alamat file hasil kompresi yang diinginkan
`public static String readOutputGIFPath(Scanner scanner)`	Menerima input alamat file gif visualisasi proses kompresi

`public static boolean readOptionGIF(Scanner scanner)`	Menerima input pengguna terkait pilihan membuat gif atau tidak
`public static void displayWelcome()`	Menampilkan display awal program

2.2.5. Kelas Node

Kelas ini merupakan implementasi simpul pada Quadtree. Berikut adalah daftar atribut dan metode pada kelas ini:

Kelas Node

Atribut:

```
private int redValue; // nilai pixel pada kanal merah
private int greenValue; // nilai pixel pada kanal hijau
private int blueValue; // nilai pixel pada kanal biru
private final int width; //panjang kolom yang direpresentasikan oleh node
private final int height;//panjang baris yang direpresentasikan oleh node
private final Point pos; // posisi pojok kiri atas node pada matrix utama
private boolean isLeaf; // true jika node adalah daun
private Node topLeft; // anak 1
private Node topRight; // anak 2
private Node botLeft; // anak 3
private Node botRight; // anak 4
```

Metode	Penjelasan

`public Node();`
 	
`public Node(int _redValue, int _greenValue, int _blueValue, boolean _isLeaf, int _row, int _col, int _height, int _width);`
 	
`public Node (int _redValue, int _greenValue, int _blueValue, boolean _isLeaf, int _row, int _col, int _height, int _width, Node _topLeft, Node _topRight, Node _botLeft, Node _botRight)`
	Konstruktor kelas Node.

`public void setRedValue(int newValue);`
`public void setGreenValue(int newValue);`
`public void setBlueValue(int newValue;`
`public void setIsLeaf(boolean newIsLeaf);`
`public void setTopLeft(Node newTopLeft);`
`public void setTopRight(Node newTopRight);`
`public void setBotLeft(Node newBotLeft;`
`public void setBotRight(Node newBotRight);`
	Getter dan setter untuk masing-masing atribut Node.

`public int getRedValue();`
`public int getGreenValue();`
`public int getBlueValue();`
`public boolean getIsLeaf();`
`public Node getTopLeft();`
`public Node getTopRight();`
`public Node getBotLeft();`
`public Node getBotRight(){;`
`public Point getPos();`
`public int getNumRows();`
`public int getNumCols();`	

`public int getArea();`	Mengembalikan area pixel yang direpresentasikan oleh Node.

2.2.6. Kelas Point

Kelas ini dibuat untuk menunjukkan lokasi Node pada matrix utama gambar. Posisi yang disimpan adalah baris dan kolom pojok kiri atas.

Kelas Point	

Atribut:
 	
`public final int row;`
`public final int col;`	

Metode	**Penjelasan**
`public Point();` `public Point(int _row, int _col);`	Konstruktor kelas Point.

2.2.7. Kelas Quadtree

Kelas Quadtree berisi algoritma utama divide and conquer untuk kompresi gambar. Quadtree merupakan suatu model pohon yang akan terus membagi node induk menjadi empat bagian jika memenuhi kondisi-kondisi tertentu.

Kelas Quadtree

Atribut:
 	
`private final int method; // pilihan metode perhitungan error`
`private final double threshold; // ambang batas untuk pembagian quadtree`
`private final int minBlockSize; // ukuran minimum blok yang bisa dibagi`

`private int numNode; // banyaknya node pada pohon`
`private Image image; // gambar yang diproses oleh Quadtree`
`private Node root; // node akar dari Quadtree`	

Metode	**Penjelasan**
`public Quadtree();` `public Quadtree(int _method, double _threshold, int _minBlockSize, Image _image);`	Konstruktor kelas Quadtree.
`public void construct(int[][] red, int[][] green, int[][] blue)`	Metode yang dipanggil kelas Main untuk mulai melakukan konstruksi Quadtree dari matriks RGB gambar. Menghasilkan akar Quadtree dan di-set ke atribut.
`public Node checker(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int h, int w)`	Metode berisi algoritma utama *divide and conquer* dalam konstruksi Quadtree. Pembuatan node-node dilakukan pada metode ini.
`public boolean isDivide(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int h, int w, int avgRed, int avgGreen, int avgBlue)`	Metode untuk megecek apakah suatu node dapat dibagi lagi menjadi empat bagian. Metode ini dapat memanggil berbagai metode pengukuran error, bergantung masukan.
`public int avgPixel(int[][] matrix, int rowTL, int colTL, int h, int w)`	Metode untuk menghitung nilai rata-rata pixel pada suatu area matrix.
`public static int getDepth(Node root)`	Metode untuk menghitung kedalaman Quadtree.
`public static int findMax(int hTL, int hTR, int hBL, int hBR)`	Metode helper untuk menentukan nilai maksimum diantara empat bilangan.

`public int getNumberOfNode();`
`public Image getImage();`
`public Node getRoot();`
	Getter dan setter atribut Quadtree.

2.2.8. Kelas Tuple2

Kelas ini merupakan kelas helper yang dapat menyimpan dua item dalam satu tuple. Kelas ini digunakan dalam salah satu metode perhitungan error, yaitu Entropy, untuk optimasi kompleksitas program.

Kelas Tuple2	

Atribut:
 	
`private T1 item1;`
`private T2 item2;`	

Metode	**Penjelasan**
`public Tuple2(T1 _item1, T2 _item2)`	Konstruktor kelas Tuple2.

`public T1 getItem1();`
`public T2 getItem2();`
 	
`public void setItem1(T1 _item1);`
`public void setItem2(T2 _item2);`
	Getter dan setter atribut Tuple2.

2.2.9. Kelas TargettedCompressor

Kelas ini berisi program untuk mengkompresi gambar dengan target persentase kompresi tertentu. Berikut atribut dan metode yang terdapatpada kelas TargettedCompressor.

Kelas TargettedCompressor	

Atribut:
 	
`private static final double INITIAL_THRESHOLD = 10.0`
`private static final int MAX_ITERATIONS = 20`
`private static final double TOLERANCE = 0.001`
`private static double THRESHOLD_TOLERANCE = 0.01`	

Metode	**Penjelasan**
`public static double compressWithTargetRatio(String imagePath, String outputPath, int errorMethod, int minBlockSize, double targetCompression)`	Mengkompresi file gambar dengan mencari threshold yang optimal untuk memenuhi target persentase.

2.2.10. Kelas Entropy

Konsep entropi informasi menggambarkan seberapa banyak keacakan atau ketidakpastian yang terdapat dalam sinyal atau gambar. Metode perhitungan error ini menggunakan probabilitas untuk menentukan nilai entropi masing-masing kanal.

Entropy	$H_c = -\sum_{i=1}^{N} P_c(i) log_2(P_c(i))$		
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$		
	H_c	=	Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
	$P_c(i)$	=	Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

Kelas Entropy	
Metode	**Penjelasan**
`public static double computeEntropy(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int h, int w)`	Metode ini menghitung rata-rata entropi dari tiga kanal warna (merah, hijau, dan biru) pada suatu bagian matriks.
`public static double computeEntropyCanal(int[][] matrix, int rowTL, int colTL, int h, int w)`	Metode ini menghitung entropi untuk satu kanal warna pada suatu bagian matriks.

2.2.11. Kelas MAD

Kelas ini digunakan untuk menghitung Mean Absolute Deviation (MAD) dari nilai piksel dalam gambar atau bagian gambar. MAD mengukur rata-rata perbedaan absolut antara setiap nilai piksel dan nilai rata-rata seluruh piksel dalam area yang dianalisis. Pada program ini, *mean absolute deviation* ini digunakan sebagai salah satu *error measurement method* dalam kompresi gambar.

Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N}\sum_{i=1}^{N} \lvert P_{i,c} - \mu_c \rvert$		
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$		
	MAD_c	=	Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$	=	Nilai piksel pada posisi *i* untuk kanal warna *c*
	μ_c	=	Nilai rata-rata tiap piksel dalam satu blok
	N	=	Banyaknya piksel dalam satu blok

Kelas MAD	
Metode	**Penjelasan**
`public static double computeMAD(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int h, int w)`	Metode ini menghitung rata-rata MAD dari tiga kanal warna (merah, hijau, dan biru) pada suatu bagian matriks.
`public static double computeMADCanal(int[][] matrix, int rowTL, int colTL, int h, int w)`	Metode ini menghitung MAD untuk satu kanal warna pada suatu bagian matriks.

2.2.12. Kelas MPD

Kelas ini berisi metode-metode yang digunakan untuk menghitung perbedaan maksimum antara nilai piksel dalam sebuah gambar atau bagian gambar (submatrix). Perbedaan maksimum ini digunakan sebagai salah satu *error measurement method* dalam kompresi gambar.

Max Pixel Difference	$D_c = max(P_{i,c}) - min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok $P_{i,c}$ = Nilai piksel pada posisi *i* untuk channel warna *c*

Kelas MPD	
Metode	**Penjelasan**
`public static double computeMPD(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int w, int h)`	Metode ini menghitung rata-rata MPD dari tiga kanal warna (merah, hijau, dan biru) pada suatu bagian matriks.
`public static double computeMPDCanal(int[][] matrix, int rowTL, int colTL, int w, int h)`	Metode ini menghitung MPD untuk satu kanal warna pada suatu bagian matriks.
`private static int[] findMinMax(int[][] matrix, int startRow, int startCol, int endRow, int endCol)`	Metode ini berfungsi untuk mendapatkan nilai minimum dan maksimum dalam

	submatrix dengan menggunakan pendekatan divide and conquer.

2.2.13. Kelas SSIM

Structural Similarity Index (SSIM) merupakan metode perhitungan error dengan mempertimbangkan tiga aspek, yaitu *luminance,* kontras, dan struktur. Metode ini menggunakan parameter rata-rata dan standar deviasi.

[Bonus] Structural Similarity Index (SSIM) (Referensi tambahan)	$SSIM_c(x,y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

Kelas SSIM	
Metode	**Penjelasan**
`public static double computeSSIM(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int h, int w, int avgRed, int avgGreen, int avgBlue)`	Metode ini menghitung nilai SSIM RGB dengan beban *(weight)* tertentu untuk masing-masing kanal warna (merah, hijau, dan biru) pada suatu bagian matriks.
`public static double computeSSIMCanal(int[][] matrix, int rowTL, int colTL, int h, int w, int avgColor)`	Metode ini menghitung SSIM untuk satu kanal warna pada suatu bagian matriks.

2.2.14. Kelas Variance

Kelas ini digunakan untuk menghitung variansi (variance) dari nilai piksel dalam gambar atau bagian gambar tertentu. Variansi merupakan ukuran seberapa jauh sebuah kumpulan nilai tersebar dari nilai rata-ratanya. Variansi ini digunakan sebagai salah satu *error measurement method* dalam kompresi gambar.

Kelas Variance

Metode	Penjelasan
`public static double computeVariance(int[][] red, int[][] green, int[][] blue, int rowTL, int colTL, int w, int h)`	Metode ini menghitung rata-rata variansi dari tiga kanal warna (merah, hijau, dan biru) pada bagian tertentu dari matriks.
`public static double computeVarianceCanal(int[][] matrix, int rowTL, int colTL, int w, int h)`	Metode ini menghitung variansi untuk satu kanal warna pada suatu bagian matriks
`private static long sum(int[][] matrix, int startRow, int startCol, int endRow, int endCol)`	Metode ini menghitung jumlah semua nilai dalam submatrix dengan pendekatan divide-and-conquer
private static double helperVariance(int[][] matrix, double mean, int startRow, int startCol, int endRow, int endCol)	Metode ini menghitung jumlah selisih kuadrat antara setiap nilai dalam submatrix dan nilai mean.

BAB III

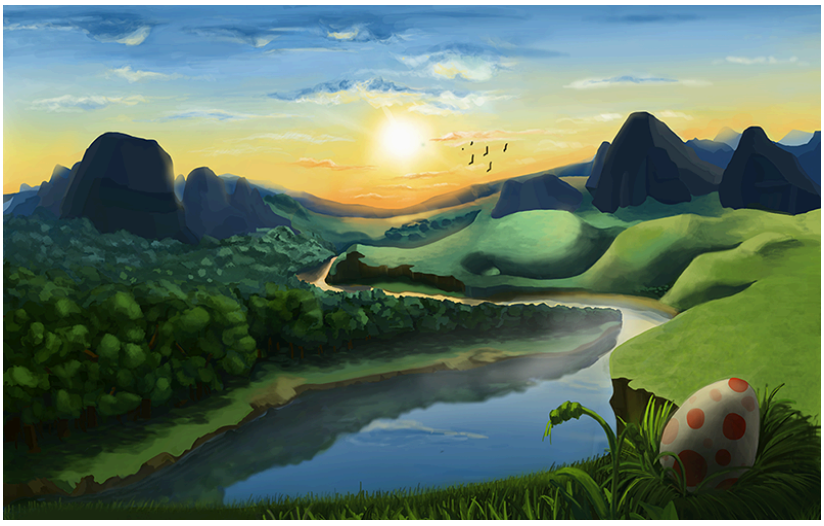
EKSPERIMEN DAN ANALISIS

3.1. Test Case 1

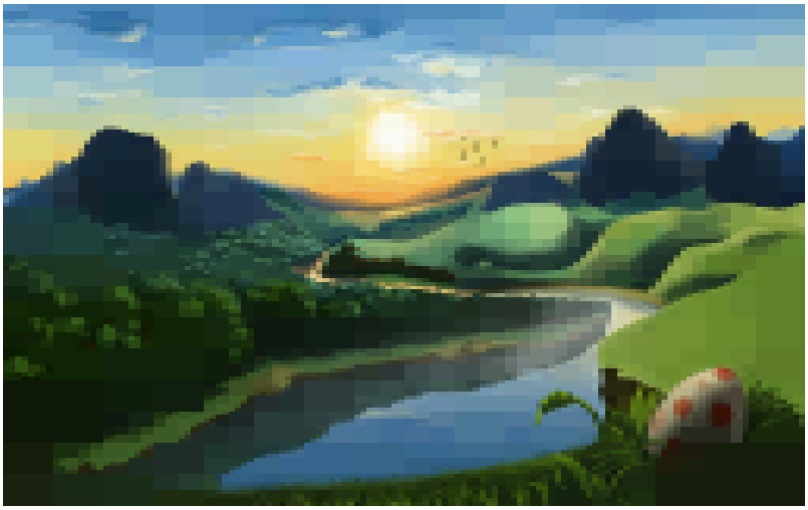
Menggunakan variance sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 1
============================================================
Masukkan nilai ambang batas (threshold): 100
============================================================
Masukkan ukuran blok minimum: 10
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed1.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 132 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 44332 bytes
Persentase kompresi: 91.19%
Kedalaman pohon: 7
Banyak simpul pada pohon: 7645
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed1.png
============================================================
Mau membuat GIF? (y/n) n
============================================================
```

- Image Asli:



- Image Hasil Kompresi:



3.2. Test Case 2

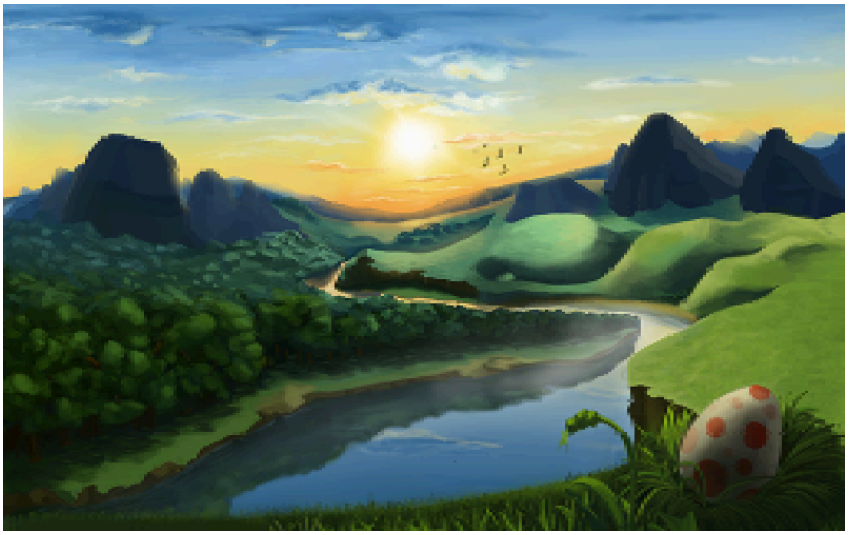
Menggunakan variance sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 1
============================================================
Masukkan nilai ambang batas (threshold): 0.1
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed2.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 154 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 172006 bytes
Persentase kompresi: 65.81%
Kedalaman pohon: 8
Banyak simpul pada pohon: 79717
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed2.png
============================================================
Mau membuat GIF? (y/n)
```

-

- Image Asli:

- Image Hasil Kompresi:

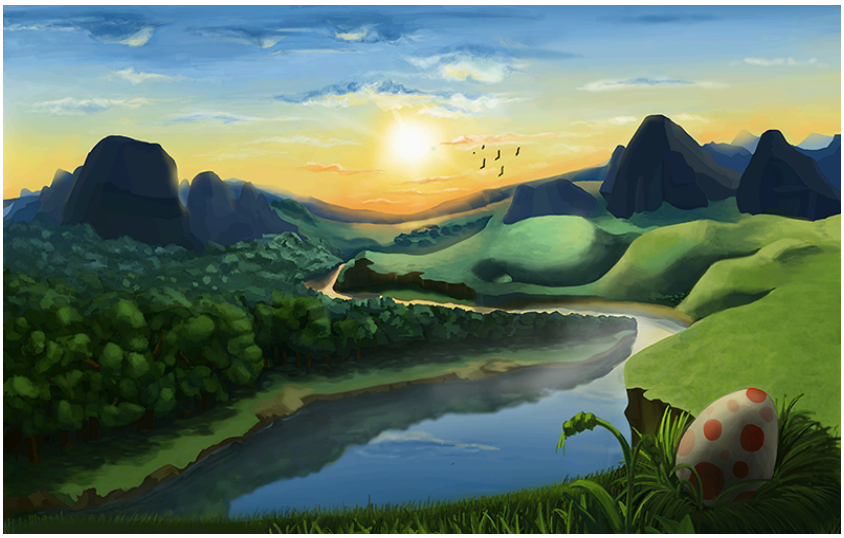


3.3. Test Case 3

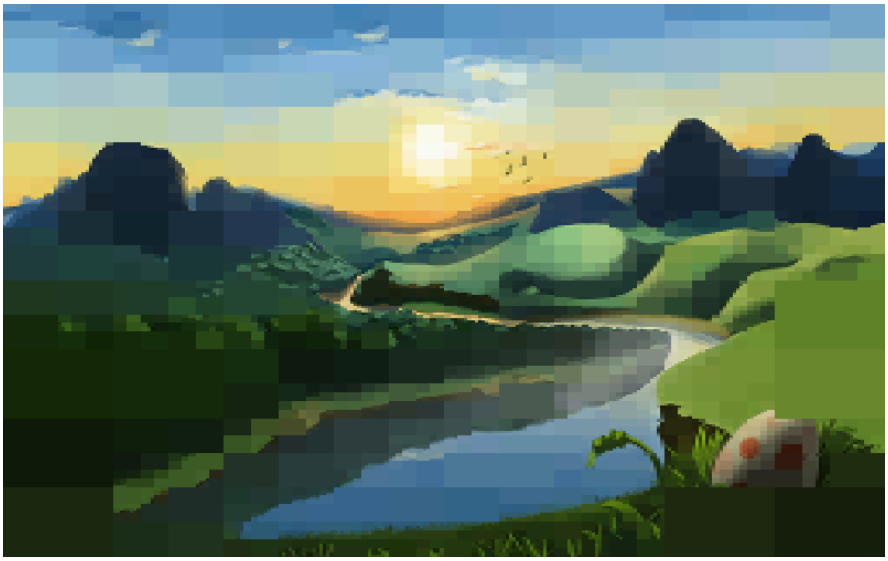
Menggunakan *mean absolute deviation* (MAD) sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 2
============================================================
Masukkan nilai ambang batas (threshold): 10
============================================================
Masukkan ukuran blok minimum: 4
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed3.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 131 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 53185 bytes
Persentase kompresi: 89.43%
Kedalaman pohon: 8
Banyak simpul pada pohon: 9485
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed3.png
============================================================
```

- Image Asli:



- Image Hasil Kompresi:



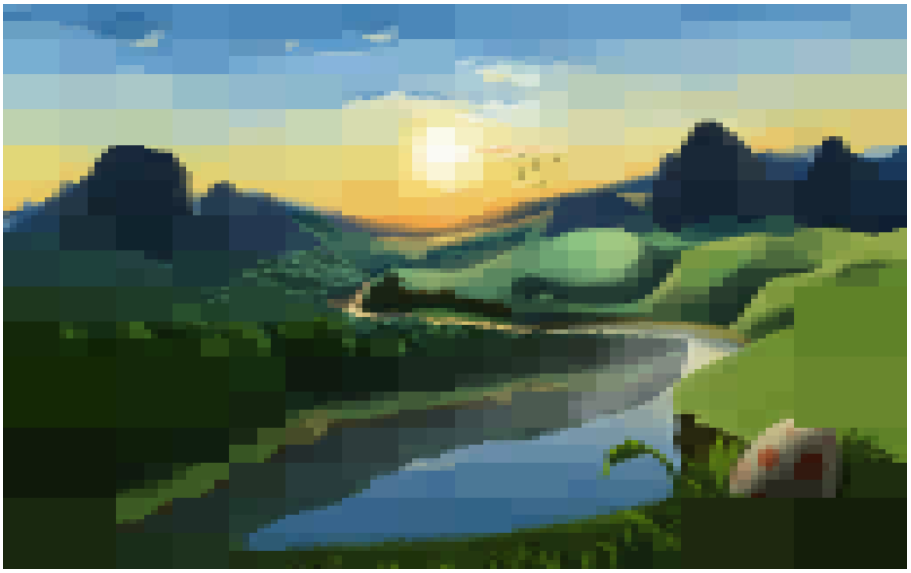
3.4. Test Case 4

Menggunakan *mean absolute deviation* (MAD) sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 2
============================================================
Masukkan nilai ambang batas (threshold): 10
============================================================
Masukkan ukuran blok minimum: 7
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed4.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 131 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 37869 bytes
Persentase kompresi: 92.47%
Kedalaman pohon: 7
Banyak simpul pada pohon: 4889
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed4.png
============================================================
Mau membuat GIF? (y/n) n
============================================================
```

- Image Asli:

- Image Hasil Kompresi:

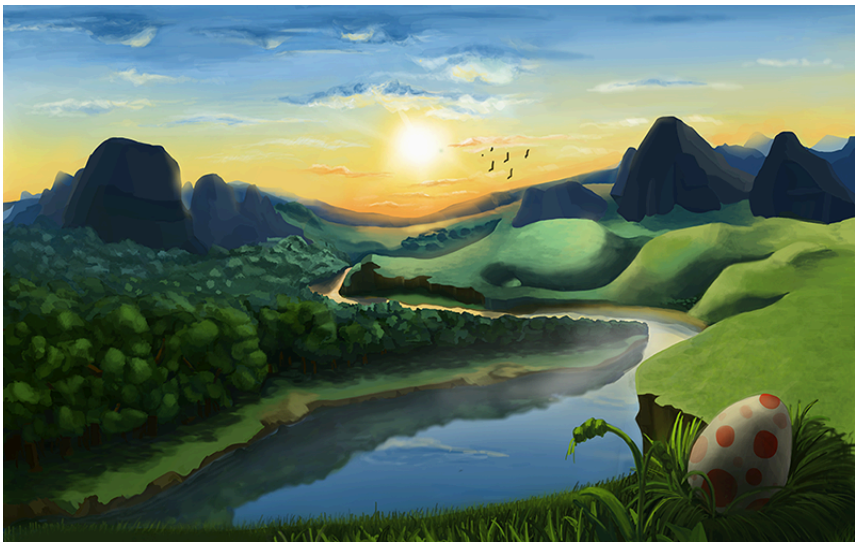


3.5. Test Case 5

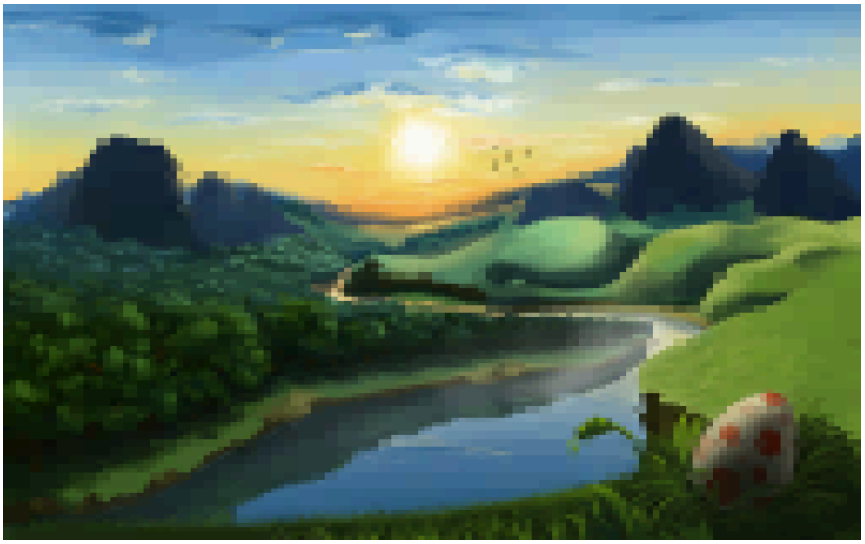
Menggunakan *mean absolute deviation* (MAD) sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 3
============================================================
Masukkan nilai ambang batas (threshold): 18
============================================================
Masukkan ukuran blok minimum: 20
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed5.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 161 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 57391 bytes
Persentase kompresi: 88.59%
Kedalaman pohon: 7
Banyak simpul pada pohon: 13413
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed5.png
============================================================
Mau membuat GIF? (y/n) n
============================================================
```

- Image Asli:



- Image Hasil Kompresi:



3.6. Test Case 6

Menggunakan *max pixel difference* (MPD) sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 3
============================================================
Masukkan nilai ambang batas (threshold): 7
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed6.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 194 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 152493 bytes
Persentase kompresi: 69.69%
Kedalaman pohon: 8
Banyak simpul pada pohon: 60109
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed6.png
============================================================
```

- Image Asli:

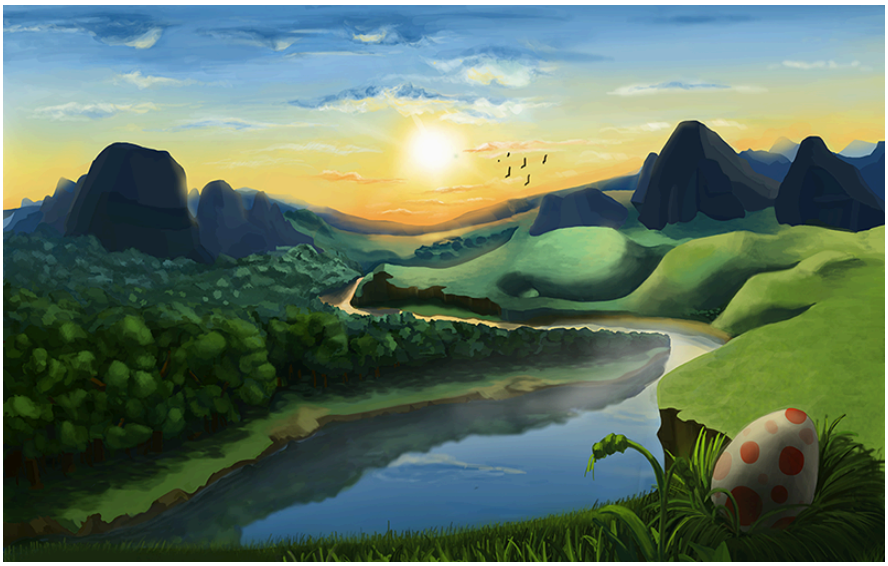
- Image Hasil Kompresi:

3.7. Test Case 7

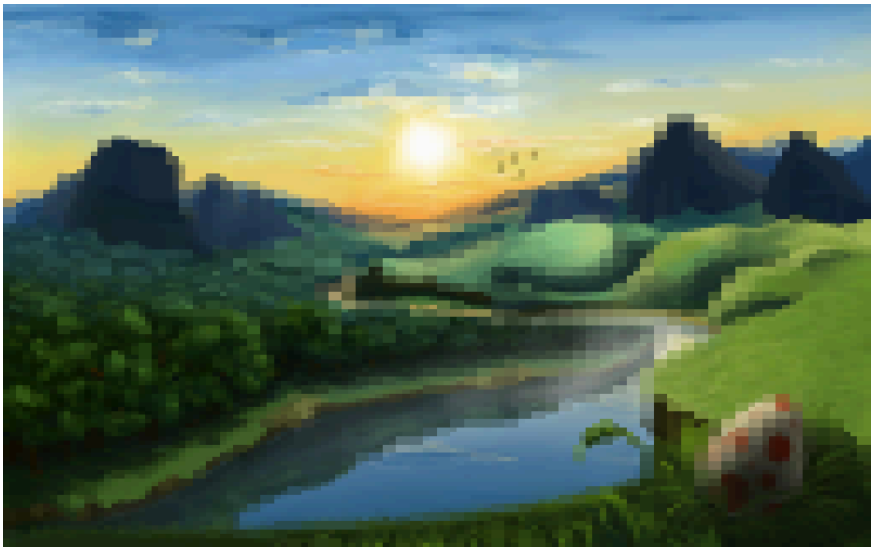
Menggunakan entropy sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 4
============================================================
Masukkan nilai ambang batas (threshold): 20
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed7.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 2296 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 55614 bytes
Persentase kompresi: 88.95%
Kedalaman pohon: 7
Banyak simpul pada pohon: 18649
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed7.png
============================================================
Mau membuat GIF? (y/n)
```

- Image Asli:



- Image Hasil Kompresi:

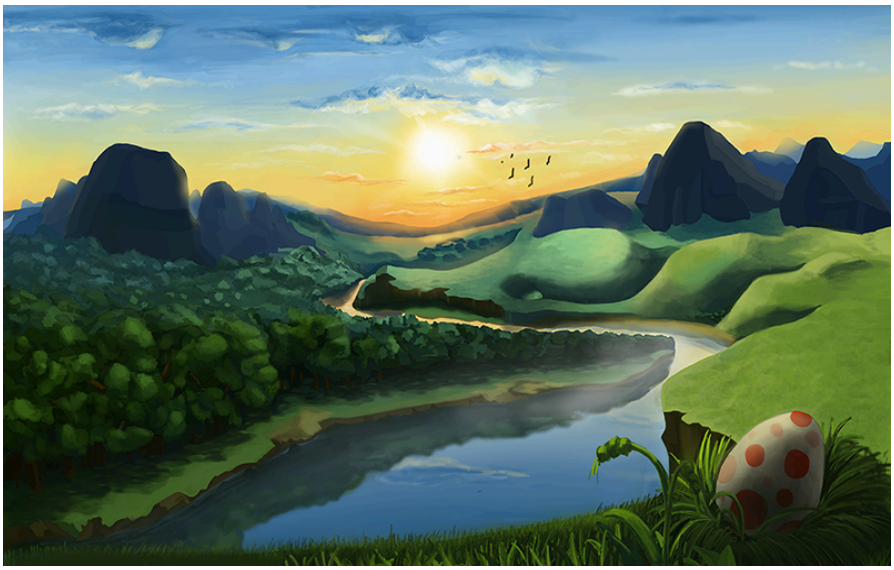


3.8. Test Case 8

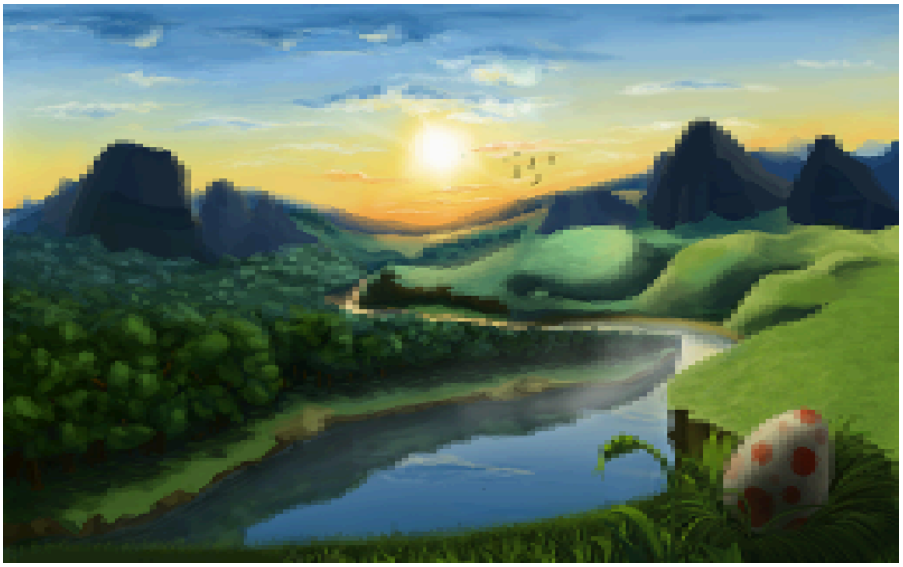
Menggunakan entropy sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 4
============================================================
Masukkan nilai ambang batas (threshold): 7
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed8.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 2382 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 156107 bytes
Persentase kompresi: 68.97%
Kedalaman pohon: 8
Banyak simpul pada pohon: 70677
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed8.png
============================================================
Mau membuat GIF? (y/n)
```

- Image Asli:



- Image Hasil Kompresi

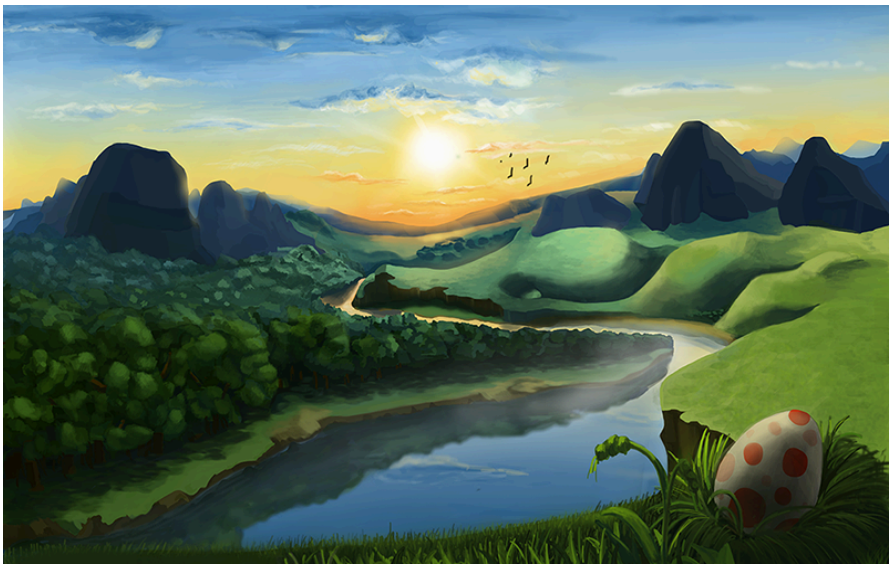


3.9. Test Case 9

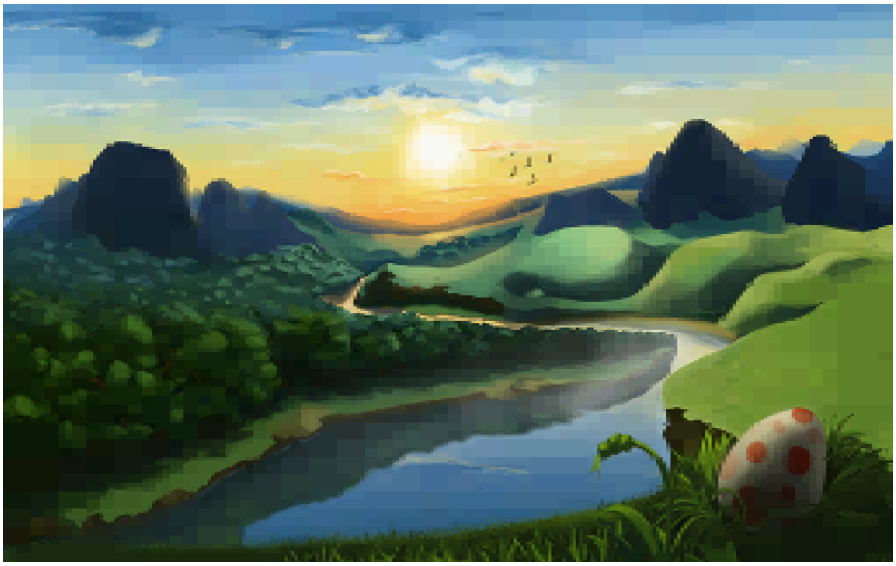
Bonus Target Kompresi

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 3
============================================================
Masukkan nilai ambang batas (threshold): 3
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0.82
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed9.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 970 ms
Threshold yang digunakan: 25.79
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 90628 bytes
Persentase kompresi: 81.99%
Kedalaman pohon: 8
Banyak simpul pada pohon: 25517
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed9.png
============================================================
```

- Image Asli:



- Image Hasil Kompresi



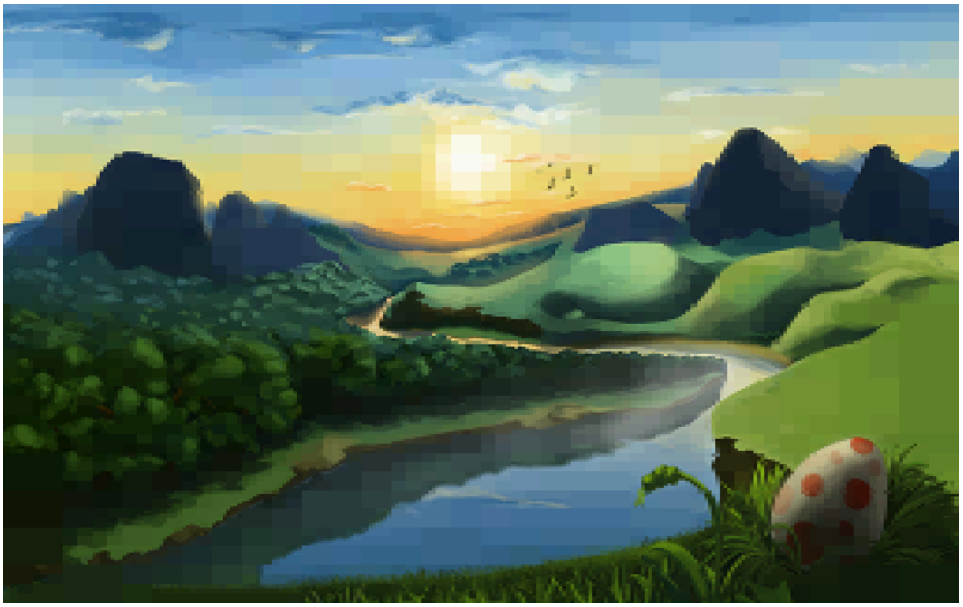
3.10. Test Case 10

Bonus SSIM sebagai error measurement method.

```
============================================================
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\USER\Downloads\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 5
============================================================
Masukkan nilai ambang batas (threshold): 0.4
============================================================
Masukkan ukuran blok minimum: 5
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed10.png

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 159 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 95893 bytes
Persentase kompresi: 80.94%
Kedalaman pohon: 8
Banyak simpul pada pohon: 27521
Gambar hasil kompresi disimpan di:
C:\Users\USER\Documents\Nayla\Tucil2_13523043_13523079\test\compressed10.png
============================================================
Mau membuat GIF? (y/n)
```

- Image Asli:

- Image Hasil Kompresi

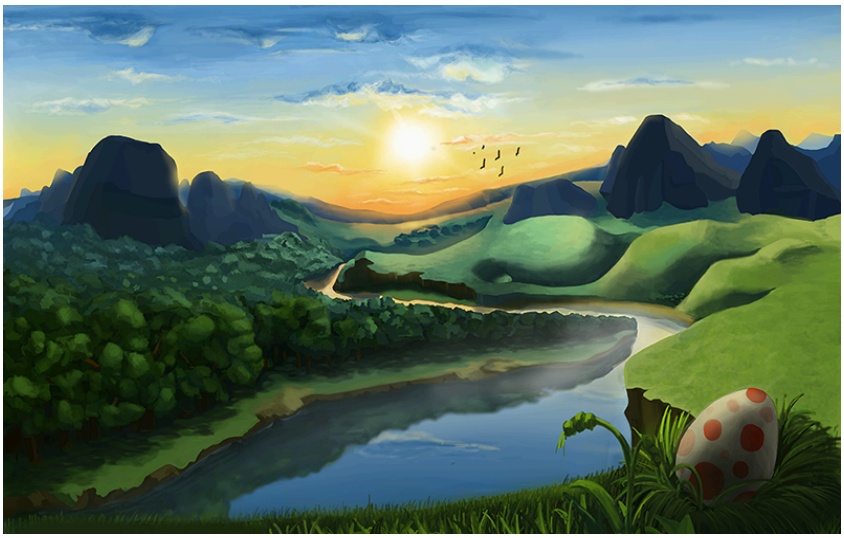


3.11. Test Case 11

Bonus GIF

```
                    PROGRAM KOMPRESI GAMBAR
============================================================


Masukkan alamat absolut file gambar yang ingin dikompresi:
C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main\test\testcase.png
============================================================
Metode perhitungan error yang tersedia:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index
Masukkan pilihan Anda: 3
============================================================
Masukkan nilai ambang batas (threshold): 15
============================================================
Masukkan ukuran blok minimum: 4
============================================================
Masukkan target persentase kompresi (0.0-1.0, 0 untuk menonaktifkan): 0
============================================================
Masukkan alamat absolut untuk output file hasil kompresi: C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main\test\c
ompressed11.jpg

============================================================
                    HASIL KOMPRESI
============================================================
Waktu eksekusi: 285 ms
Ukuran gambar asli: 503111 bytes
Ukuran gambar terkompresi: 46726 bytes
Persentase kompresi: 90,71%
Kedalaman pohon: 8
Banyak simpul pada pohon: 40257
Gambar hasil kompresi disimpan di:
C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main\test\compressed11.jpg
============================================================
Mau membuat GIF? (y/n) y
============================================================
Masukkan alamat absolut untuk output file gif: C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main\test\compressed11
.gif
GIF telah dibuat!
Letak file: C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main\test\compressed11.gif
============================================================
PS C:\Users\ASUS\Downloads\Tucil2_13523043_13523079-main\Tucil2_13523043_13523079-main>
```

- Image Asli



- Hasil GIF

BAB IV
ANALISIS ALGORITMA

4.1. Analisis Kompleksitas Program

4.1.1. Analisis Kompleksitas Algoritma Divide and Conquer Utama (Proses Kompresi)

Algoritma kompresi gambar yang diimplementasikan menggunakan pendekatan divide and conquer dengan struktur data Quadtree bekerja dengan membagi gambar menjadi empat kuadran secara rekursif. Pada setiap tahap, algoritma mengevaluasi apakah blok perlu dibagi lebih lanjut berdasarkan kriteria pengukuran error yang ditentukan, dan proses akan berhenti ketika ukuran blok mencapai batas minimum atau nilai error berada di bawah threshold. Fungsi utama checker adalah melakukan pembagian rekursif dengan kompleksitas $O(h \times w)$ pada setiap pemanggilan untuk menghitung rata-rata pixel dan mengevaluasi kriteria pembagian.

Analisis secara rekursif menunjukkan bahwa untuk gambar berukuran $n \times n$, kompleksitas di setiap level tetap $O(n^2)$, dengan jumlah level maksimal mencapai $\log_2(n)$. Pada level 0 (root), terdapat satu blok berukuran $n \times n$, level 1 terdapat empat blok berukuran $n/2 \times n/2$, dan pola ini berlanjut hingga level terdalam.

Kompleksitas waktu algoritma ini dalam kasus terburuk adalah $O(n^2 \times \log_2(n))$ ketika hampir semua blok perlu dibagi hingga mencapai ukuran minimum, sementara dalam kasus terbaik dapat mencapai $O(n^2)$ ketika gambar sangat homogen dan tidak memerlukan banyak pembagian. Berbagai faktor seperti karakteristik gambar, parameter threshold, dan ukuran blok minimum mempengaruhi kompleksitas akhir, namun secara keseluruhan algoritma ini menunjukkan efisiensi yang jauh lebih baik dibandingkan pendekatan naif yang mengevaluasi semua kemungkinan kombinasi blok dengan kompleksitas $O(n^4)$.

4.1.2. Analisis Kompleksitas Perhitungan Variance

Dalam metode `computeVarianceCanal`, algoritma melakukan dua operasi utama: menghitung jumlah (sum) nilai dalam matriks melalui fungsi sum, dan menghitung jumlah selisih kuadrat melalui fungsi `helperVariance`. Kedua fungsi ini mengimplementasikan pendekatan divide and conquer, di mana area matriks dibagi menjadi empat bagian secara rekursif.

Analisis kompleksitas fungsi `sum` dan `helperVariance` memiliki pola rekursif yang identik:

- Basis: jika area yang diproses hanya 1 elemen, waktu konstan O(1)
- Rekursi: area dibagi menjadi 4 sub-area yang lebih kecil
- Kompleksitas rekursif: T(n) = 4T(n/4) + O(1)

Menggunakan teorema master, karena a = 4, b = 4, dan f(n) = O(1), maka f(n) = O(n^(log b(a))). Dalam hal ini, log 4(4) = 1, sehingga kompleksitas adalah O(n^1) = O(n) di mana n adalah jumlah elemen dalam area yang diproses.

Sehingga, dapat disimpulkan bahwa penggunaan metode Variance tidak mengubah kompleksitas asimptotik dari algoritma Quadtree, yaitu tetap $O(n^2 \times \log_2(n))$ pada kasus terburuk

4.1.3. Analisis Kompleksitas Perhitungan Mean Absolute Deviation (MAD)

Metode `computeMAD` memanggil fungsi `computeMADCanal` untuk ketiga kanal warna (merah, hijau, dan biru) dan mengembalikan rata-ratanya. Fungsi `computeMADCanal` melakukan iterasi melalui seluruh piksel dalam blok gambar dengan dua loop bersarang untuk menghitung jumlah nilai piksel dan menyimpannya dalam array datar. Selanjutnya, metode ini melakukan iterasi kedua melalui array datar untuk menghitung jumlah deviasi absolut terhadap nilai rata-rata. Kedua iterasi ini memiliki kompleksitas $O(h \times w)$, di mana h dan w adalah tinggi dan lebar blok gambar.

Pengaruh penggunaan metode MAD terhadap kompleksitas waktu keseluruhan algoritma kompresi gambar tetap konsisten dengan analisis sebelumnya. Meskipun implementasi internal berbeda, kompleksitas asimptotik MAD tetap $O(h \times w)$ untuk setiap blok, sama seperti metode Variance. Dengan demikian, kompleksitas waktu keseluruhan algoritma tetap $O(n^2 \times \log_2(n))$ pada kasus terburuk dan $O(n^2)$ pada kasus terbaik, di mana n adalah dimensi gambar.

4.1.4. Analisis Kompleksitas Perhitungan Max Pixel Difference (MPD)

Metode MPD memperhitungkan selisih antara nilai piksel maksimum dan minimum dalam suatu blok gambar. Fungsi `computeMPD` memanggil metode `computeMPDCanal` untuk masing-masing kanal warna, yang kemudian memanggil

fungsi rekursif `findMinMax` untuk mencari nilai minimum dan maksimum dalam area yang ditentukan.

Analisis kompleksitas waktu untuk MPD berfokus pada fungsi `findMinMax` yang mengimplementasikan pendekatan divide and conquer. Fungsi ini membagi area matriks menjadi empat bagian dan mencari nilai minimum dan maksimum secara rekursif. Menggunakan analisis rekursif, kompleksitas fungsi `findMinMax` adalah $T(n) = 4T(n/4) + O(1)$, yang berdasarkan teorema master menghasilkan kompleksitas $O(n)$ di mana n adalah jumlah elemen dalam area ($h \times w$).

Dengan demikian, kompleksitas waktu dari metode MPD adalah $O(h \times w)$, sama seperti metode Variance dan MAD. Akibatnya, penggunaan metode MPD tidak mengubah kompleksitas keseluruhan algoritma kompresi gambar, yang tetap $O(n^2 \times \log_2(n))$ dalam kasus terburuk dan $O(n^2)$ dalam kasus terbaik.

4.1.5. Analisis Kompleksitas Program Kompresi dengan Entropy

Metode ini menghitung entropi Shannon dari distribusi nilai piksel dalam suatu blok gambar. Fungsi `computeEntropy` memanggil metode `computeEntropyCanal` untuk ketiga kanal warna dan mengembalikan rata-ratanya. Pada metode `computeEntropyCanal`, algoritma menggunakan dua loop bersarang untuk menghitung frekuensi kemunculan setiap nilai piksel unik dalam blok, yang kemudian digunakan untuk menghitung entropi.

Analisis kompleksitas waktu untuk metode Entropy menunjukkan pola yang berbeda dari metode-metode sebelumnya. Loop pertama memiliki kompleksitas $O(h \times w \times u)$, dimana u adalah jumlah nilai piksel unik, karena untuk setiap piksel algoritma melakukan pencarian linear pada daftar nilai unik. Loop kedua juga memiliki kompleksitas $O(h \times w \times u)$ untuk menghitung entropi. Dalam kasus terburuk dimana semua piksel memiliki nilai berbeda, $u = h \times w$, sehingga kompleksitas totalnya bisa mencapai $O((h \times w)^2)$.

Akibatnya, penggunaan metode Entropy dapat meningkatkan kompleksitas algoritma kompresi gambar menjadi $O(n^4)$ dalam kasus terburuk, meskipun dalam praktiknya biasanya jauh lebih baik karena jumlah nilai piksel unik cenderung jauh lebih kecil dari total piksel.

4.1.6. Analisis Kompleksitas Program Kompresi dengan SSIM

Metode SSIM ini membandingkan kemiripan struktural antara blok gambar dengan representasi rata-ratanya, mempertimbangkan tiga komponen: luminance, contrast, dan structure. Fungsi `computeSSIM` memanggil metode `computeSSIMCanal` untuk masing-masing kanal warna dengan bobot berbeda (0.30 untuk merah, 0.59 untuk hijau, dan 0.11 untuk biru), mencerminkan sensitivitas persepsi manusia terhadap warna.

Dari segi kompleksitas waktu, metode SSIM menggunakan pendekatan iteratif langsung seperti MAD. Dalam fungsi `computeSSIMCanal`, terdapat loop ganda untuk mengumpulkan nilai piksel dan menghitung jumlahnya dengan kompleksitas $O(h \times w)$, diikuti dengan loop tunggal untuk menghitung variance dengan kompleksitas $O(h \times w)$ lagi. Operasi matematika lainnya memiliki kompleksitas konstan. Dengan demikian, kompleksitas waktu total untuk metode SSIM adalah $O(h \times w)$.

Penggunaan metode SSIM tidak mengubah kompleksitas asimptotik keseluruhan algoritma kompresi gambar, tetap $O(n^2 \times \log_2(n))$ dalam kasus terburuk dan $O(n^2)$ dalam kasus terbaik.

BAB V

IMPLEMENTASI BONUS

5.1. Target Persentase Kompresi

Fitur target persentase kompresi memungkinkan pengguna untuk menentukan secara spesifik berapa persentase kompresi yang ingin dicapai pada gambar yang diproses. Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai threshold secara otomatis untuk mencapai target persentase kompresi yang ditentukan.

Pada implementasinya, fitur ini menggunakan algoritma pencarian biner (binary search) untuk mencari nilai threshold optimal yang akan menghasilkan persentase kompresi yang mendekati target yang diinginkan pengguna. Algoritma ini dimulai dengan menginisialisasi batas atas dan batas bawah untuk nilai threshold berdasarkan error measurement method yang digunakan. Kemudian, algoritma ini melakukan iterasi dengan jumlah maksimal 20, untuk mencoba kompresi gambar hingga sesuai dengan target kompresi yang diinginkan. Toleransi yang digunakan dalam perhitungan persentase kompresi adalah 0.00001 atau 0.01%.

Pada setiap iterasi, algoritma ini akan memperkecil range pencarian threshold untuk iterasi berikutnya. Hal ini dilakukan dengan mengganti nilai batas atas dan batas bawah threshold sesuai dengan rasio kompresi yang dihasilkan. Jika rasio kompresi yang dihasilkan lebih tinggi dari target, pada iterasi berikutnya akan dicoba nilai threshold yang lebih rendah. Sebaliknya, jika rasio kompresi yang dihasilkan lebih rendah dari target, maka pada iterasi berikutnya akan dicoba nilai threshold yang lebih tinggi.

Namun, target kompresi tidak selalu dapat dicapai. Hal ini terjadi karena adanya faktor-faktor lain yang ikut mempengaruhi proses kompresi, seperti *error measurement method* yang digunakan. Metode pengukuran error seperti Entropy cenderung kurang akurat dalam beberapa situasi karena kompleksitas perhitungannya yang tinggi dan sensitivitasnya terhadap nilai piksel unik. Selain itu, metode-metode seperti MAD dan MPD juga dapat memberikan hasil yang tidak akurat pada gambar dengan distribusi nilai piksel yang sangat tidak merata atau memiliki outlier signifikan, karena mereka tidak mempertimbangkan frekuensi kemunculan nilai piksel dalam perhitungannya.

Selain hasil yang terkadang tidak akurat, fitur target kompresi ini juga mengalami peningkatan kompleksitas waktu yang signifikan ketika menggunakan metode pengukuran error Entropy. Kompleksitas algoritma dapat mencapai $O(n^4)$ karena setiap iterasi dalam pencarian biner mengharuskan rekonstruksi Quadtree disertai dengan perhitungan Entropy yang memerlukan evaluasi distribusi probabilitas nilai piksel pada setiap region. Hal ini mengakibatkan beban komputasi yang jauh lebih tinggi dibandingkan dengan metode pengukuran error alternatif yang tersedia dalam sistem.

5.2. Structural Similarity Index (SSIM)

Structural Similarity Index adalah metode pengukuran error dengan mempertimbangkan tiga aspek, yaitu *luminance*, kontras, dan struktur pada gambar sebelum dan setelah kompresi. Pada implementasinya, rata-rata pixel gambar pada suatu node dibandingkan dengan node induk *(parent)*. Faktor luminance, kontras dan struktur dikalkulasi dan didapat fungsi SSIM untuk nilai pada node saat ini dan node induk (nilai pixel setelah kompresi) adalah sebagai berikut:

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

Pada formula tersebut, terdapat tujuh parameter yang mempengaruhi nilai SSIM, yaitu rata-rata pixel dan variansi sebelum kompresi, rata-rata pixel dan variansi setelah kompresi, kovarians, konstanta 1, dan konstanta 2. Oleh karena variansi Y dan kovarians bernilai 0 akibat nilai kompresi yang seragam, peran konstanta C1 dan C2 disini sangatlah penting. Wang, Zhou, et.all menjelaskan bahwa nilai C didapat dari perhitungan $C_n = (K_n L)^2$ dengan L adalah *dynamic range* dari pixel, yakni 255 dan $K_1 = 0.01$ dan $K_2 = 0.03$.

Nilai akhir SIM untuk seluruh kanal merah, kuning, dan biru ditentukan dengan bobot *relative luminance* w_c dari masing-masing kanal c. Bobot untuk kanal merah adalah $w_R = 0.299$, kanal biru $w_B = 0.114$, dan kanal hijau $w_G = 0.587$. Implementasi lebih detail dapat dilihat pada deskripsi kelas pada bagian 2.2.12.

5.3. GIF Proses Kompresi

Proses kompresi gambar diilustrasikan melalui animasi frame-frame yang dikonstruksi sebagai GIF. Pada program ini, pembentukan GIF dilakukan setelah pemrosesan kompresi gambar dengan Quadtree. Daftar dan tanggung jawab masing-masing metode yang digunakan dalam implementasi ini dapat dilihat pada bagian 2.2.2.

Program akan membaca node pada masing-masing lapisan kedalaman dan menyimpannya ke dalam suatu List berisi List node. Kemudian, program akan melakukan iterasi untuk membentuk gambar (frame) di setiap lapisan kedalaman. Setelah itu, matriks berisi RGB akan diperbarui seiring bertambahnya kedalaman pohon yang dikonstruksi. Iterasi dilakukan dimulai dari root hingga kedalaman pohon maksimum, sehingga frame yang terbentuk adalah sejumlah kedalaman pohon. Frame dibuat dalam format .png dan disimpan secara sementara pada folder /test/frames. Setelah seluruh lapisan dikonstruksi, program akan menghapus frame yang telah dibuat. Ekstraksi

matriks menjadi gambar untuk setiap frame dipilih agar komputer tidak terlalu berat menyimpan BufferedImage sementara.

LAMPIRAN

6.1. Tautan *repository* Github

https://github.com/najwakahanifatima/Tucil2_13523043_13523079

6.2. Tabel 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	✓	

DAFTAR PUSTAKA

Munir, Rinaldi. (2025). Algoritma Divide and Conquer - Bagian 1. Diakses pada 9 April 2025, dari: https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf.

Tsai, DY., Lee, Yongbum., Matsuyama, Eri. 2007. Information Entropy Measure for Evaluation of Image Quality. Diakses dari: https://pmc.ncbi.nlm.nih.gov/articles/PMC3043833/#:~:text=The%20concept%20of%20information%20entropy,by%20the%20signal%20or%20image.

Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios: Recommendation ITU-R BT.601-6 : (Question ITU-R 1/6). (2007). Diakses dari: RECOMMENDATION ITU-R BT.601-7 – Studio encoding parameters of digital television for standard 4:3 and wide-screen 16:9 aspect ratios

Fan, S. S., & Chuang, Y. (2013). An entropy-based method for color image registration. Taiwan, 417–421. https://doi.org/10.5220/0004210504170421.

Image quality assessment: from error visibility to structural similarity | IEEE Journals & Magazine | IEEE Xplore

https://www.w3.org/TR/AERT/#color-contrast