

**Tugas Kecil 1 IF2211 Strategi Algoritma
Semester II tahun 2022/2023**

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:
Muhammad Iqbal Haidar (13523111)

**PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
2025**

DAFTAR ISI

BAB I.....	2
DESKRIPSI MASALAH.....	3
SPESIFIKASI.....	4
BAB II.....	5
ALGORITMA.....	5
IMPLEMENTASI.....	5
BAB III.....	11
TESTING.....	12
BAB IV.....	16
ANALISIS.....	16
BAB V.....	18
KESIMPULAN.....	18
SARAN.....	18

BAB I

DESKRIPSI MASALAH



Gambar 1. Quadtree dalam Kompresi Gambar

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

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

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

SPEKIFIKASI

- 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.
- Alur program:
 1. [INPUT] **alamat absolut** gambar yang akan dikompresi.
 2. [INPUT] metode perhitungan error (gunakan penomoran sebagai input).
 3. [INPUT] ambang batas (pastikan range nilai sesuai dengan metode yang dipilih).
 4. [INPUT] ukuran block minimum.
 5. [INPUT] Target persentase kompresi (floating number, 1.0 = 100%), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
 6. [INPUT] **alamat absolut** gambar hasil kompresi.
 7. [INPUT] **alamat absolut gif** (bonus).
 8. [OUTPUT] waktu eksekusi.
 9. [OUTPUT] ukuran gambar sebelum.
 10. [OUTPUT] ukuran gambar setelah.
 11. [OUTPUT] persentase kompresi.
 12. [OUTPUT] kedalaman pohon.
 13. [OUTPUT] banyak simpul pada pohon.
 14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
 15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).
- Berkas laporan yang dikumpulkan adalah laporan dalam bentuk PDF yang setidaknya berisi:
 1. Algoritma ***divide and conquer*** yang digunakan, jelaskan langkah-langkahnya, **bukan hanya notasi pseudocode dan BUKAN IMPLEMENTASINYA melainkan algoritmanya.**
 2. Source program dalam bahasa pemrograman yang dipilih (pastikan bahwa program telah dapat dijalankan).
 3. Tangkapan layar yang memperlihatkan *input* dan *output* (minimal sebanyak 7 buah contoh).
 4. Hasil analisis percobaan algoritma ***divide and conquer*** dalam kompresi gambar dengan metode Quadtree. Analisis dilakukan dalam bentuk paragraf/poin dan minimal memuat mengenai analisis kompleksitas algoritma program yang telah dikembangkan.
 5. Penjelasan mengenai implementasi bonus jika mengerjakan.
 6. Pranala ke *repository* yang berisi kode program.

BAB II

ALGORITMA

Algoritma Divide and Conquer Quadtree

1. Input koordinat, panjang, lebar, serta hitung nilai rata-rata warna lalu simpan dalam sebuah node yang merepresentasikan state block gambar saat itu.
2. **Hitung error** sesuai dengan metode yang digunakan, Hitung ukuran block **sebelum** dibagi empat, dan ukuran block **setelah** dibagi empat.
3. Apabila nilai error masih **dias atas threshold dan** size blok **sebelum** dibagi empat \geq minimum block size **dan** size block **setelah** dibagi empat \geq minimum block size maka lanjut langkah 5. Node ini dijadikan sebagai internal node.
4. Apabila salah satu dari tiga kondisi **tidak terpenuhi** maka hentikan rekursi pada node tersebut dan jadikan sebagai leaf node.
5. Bagi gambar menjadi empat sub-block lalu ulangi langkah 1 **untuk setiap sub-block** dengan data koordinat, panjang, dan lebar yang **disesuaikan** dengan sub-block.
6. Lakukan assignment keempat node sub-block **ke** array children yang dimiliki oleh node block sebelum pembagian.

IMPLEMENTASI

Implementasi node Quadtree

```
public class QuadTreeNode {
    private int x, y; // Koordinat kiri-atas area
    private int width, height;
    private int[] avgColor; // Warna rata-rata area [R, G, B]
    private QuadTreeNode[] children; // [0]=LU, [1]=RU, [2]=LD, [3]=RD

    // Konstruktor
    public QuadTreeNode(int x, int y, int width, int height, int[] avgColor) {
        this.x = x;
        this.y = y;
        this.width = width;
        this.height = height;
        this.avgColor = avgColor;
        this.children = null;
    }

    // Setter]
    public void setChildren(QuadTreeNode[] children) {
```

```

        this.children = children;
    }

    // Getter
    public int getX() {
        return x;
    }

    public int getY() {
        return y;
    }

    public int getWidth() {
        return width;
    }

    public int getHeight() {
        return height;
    }

    public int[] getAvgColor() {
        return avgColor;
    }

    public QuadTreeNode[] getChildren() {
        return children;
    }

    public boolean isLeaf() {
        return children == null;
    }
}

```

Implementasi builder Quadtree

```

public class QuadTreeBuilder {
    private int[][][] pixels;
    private int width, height;
    private double threshold;
}

```

```

private int minBlockSize;
private int method;

public QuadTreeBuilder(ImageParser parser, int method, double threshold,
int minBlockSize) {
    this.pixels = parser.getPixels();
    this.width = parser.getWidth();
    this.height = parser.getHeight();
    this.threshold = threshold;
    this.minBlockSize = minBlockSize;
    this.method = method;
}

public QuadTreeNode build() {
    return buildRecursive(0, 0, width, height);
}

private QuadTreeNode buildRecursive(int x, int y, int w, int h) {
    int[] avg = ImageUtils.calculateAverageRGB(pixels, x, y, w, h);
    double error;

    switch (method) {
        case 1:
            error = ImageUtils.calculateVariance(pixels, x, y, w, h, avg);
            break;
        case 2:
            error = ImageUtils.calculateMAD(pixels, x, y, w, h, avg);
            break;
        case 3:
            error = ImageUtils.calculateMaxPixelDifference(pixels, x, y,
w, h);
            break;
        case 4:
            error = ImageUtils.calculateEntropy(pixels, x, y, w, h);
            break;
        default:
            throw new IllegalArgumentException("Metode error tidak
dikenali: " + method);
    }

    int w1 = w / 2, w2 = w - w1;

```

```

        int h1 = h / 2, h2 = h - h1;

        boolean split = (w1 * h1 >= minBlockSize) &&
            (w2 * h1 >= minBlockSize) &&
            (w1 * h2 >= minBlockSize) &&
            (w2 * h2 >= minBlockSize) && error >= threshold;

        QuadTreeNode node = new QuadTreeNode(x, y, w, h, avg);

        if (split) {
            QuadTreeNode[] children = new QuadTreeNode[4];
            children[0] = buildRecursive(x, y, w1, h1); // Top-left
            children[1] = buildRecursive(x + w1, y, w2, h1); // Top-right
            children[2] = buildRecursive(x, y + h1, w1, h2); // Bottom-left
            children[3] = buildRecursive(x + w1, y + h1, w2, h2); //
Bottom-right

            node.setChildren(children);
        }
        return node;
    }
}

```

Implementasi fungsi Error Method dan Average Color

```

import java.io.IOException;

public class ImageUtils {

    public static int[] calculateAverageRGB(int[][][] pixels, int x, int y,
int width, int height) {
        long sumR = 0, sumG = 0, sumB = 0;
        int totalPixel = width * height;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                sumR += pixels[i][j][0];
                sumG += pixels[i][j][1];
                sumB += pixels[i][j][2];
            }
        }
    }
}

```



```

        int avgR = (int) (sumR / totalPixel);
        int avgG = (int) (sumG / totalPixel);
        int avgB = (int) (sumB / totalPixel);

        return new int[] { avgR, avgG, avgB };
    }

    public static double calculateVariance(int[][][] pixels, int x, int y, int
width, int height, int[] avgColor) {
        double sumSqR = 0, sumSqG = 0, sumSqB = 0;
        int totalPixel = width * height;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                int r = pixels[i][j][0];
                int g = pixels[i][j][1];
                int b = pixels[i][j][2];

                sumSqR += Math.pow(r - avgColor[0], 2);
                sumSqG += Math.pow(g - avgColor[1], 2);
                sumSqB += Math.pow(b - avgColor[2], 2);
            }
        }

        double varR = sumSqR / totalPixel;
        double varG = sumSqG / totalPixel;
        double varB = sumSqB / totalPixel;

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

    public static double calculateMAD(int[][][] pixels, int x, int y, int
width, int height, int[] avgColor) {
        double sumAbsR = 0, sumAbsG = 0, sumAbsB = 0;
        int totalPixel = width * height;

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                int r = pixels[i][j][0];
                int g = pixels[i][j][1];

```

```

        int b = pixels[i][j][2];

        sumAbsR += Math.abs(r - avgColor[0]);
        sumAbsG += Math.abs(g - avgColor[1]);
        sumAbsB += Math.abs(b - avgColor[2]);
    }
}

double madR = sumAbsR / totalPixel;
double madG = sumAbsG / totalPixel;
double madB = sumAbsB / totalPixel;

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

public static double calculateMaxPixelDifference(int[][][] pixels, int x,
int y, int width, int height) {
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int r = pixels[i][j][0];
            int g = pixels[i][j][1];
            int b = pixels[i][j][2];

            minR = Math.min(minR, r);
            minG = Math.min(minG, g);
            minB = Math.min(minB, b);

            maxR = Math.max(maxR, r);
            maxG = Math.max(maxG, g);
            maxB = Math.max(maxB, b);
        }
    }

    int diffR = maxR - minR;
    int diffG = maxG - minG;
    int diffB = maxB - minB;

    return (diffR + diffG + diffB) / 3.0;
}

```

```

    }

    public static double calculateEntropy(int[][][] pixels, int x, int y, int
width, int height) {
        int totalPixel = width * height;

        int[] histR = new int[256];
        int[] histG = new int[256];
        int[] histB = new int[256];

        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                histR[pixels[i][j][0]]++;
                histG[pixels[i][j][1]]++;
                histB[pixels[i][j][2]]++;
            }
        }

        double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;

        for (int i = 0; i < 256; i++) {
            if (histR[i] > 0) {
                double p = histR[i] / (double) totalPixel;
                entropyR -= p * (Math.log(p) / Math.log(2));
            }
            if (histG[i] > 0) {
                double p = histG[i] / (double) totalPixel;
                entropyG -= p * (Math.log(p) / Math.log(2));
            }
            if (histB[i] > 0) {
                double p = histB[i] / (double) totalPixel;
                entropyB -= p * (Math.log(p) / Math.log(2));
            }
        }

        return (entropyR + entropyG + entropyB) / 3.0;
    }

```

BAB III

TESTING

1. Testing 1

Gambar Sebelum	Gambar Sesudah	CLI
		<pre>Input Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg Threshold (0-255): 2 Minimum Block Size (0-255): 5 Output Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg Output Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg === INFO KOMPRESI === 1. Nama File: 18202111 2. Ukuran Sebelum : 2082744 bytes 3. Ukuran Setelah : 199950 bytes 4. Persentase Kompresi : 94.9% 5. Waktu Proses : 12 6. Jumlah Gambar : 1 7. Jumlah Gambar di : C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test1.jpg</pre>

2. Testing 2

Gambar Sebelum	Gambar Sesudah	CLI
		<pre>Input Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg Threshold (0-255): 2 Minimum Block Size (0-255): 5 Output Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg Output Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg === INFO KOMPRESI === 1. Nama File: 18202111 2. Ukuran Sebelum : 222948 bytes 3. Ukuran Setelah : 222942 bytes 4. Persentase Kompresi : 99.9% 5. Waktu Proses : 12 6. Jumlah Gambar : 1 7. Jumlah Gambar di : C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test2.jpg</pre>

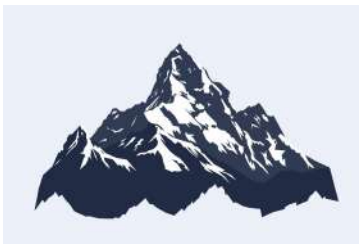

3. Testing 3

Gambar Sebelum	Gambar Sesudah	CLI
		<pre>Input Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg Threshold (0-255): 2 Minimum Block Size (0-255): 5 Output Path: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg Output Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg Error Message: C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg === INFO KOMPRESI === 1. Nama File: 18202111 2. Ukuran Sebelum : 970381 bytes 3. Ukuran Setelah : 227028 bytes 4. Persentase Kompresi : 97.6% 5. Waktu Proses : 12 6. Jumlah Gambar : 1 7. Jumlah Gambar di : C:\Users\DEBA-LAPTOP\PIF\Strategi\Foto12\18202111\test\test3.jpg</pre>



4. Testing 4

Gambar Sebelum	Gambar Sesudah	CLI
		<pre> Input Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test4.jpg Error Method (InputPath, 2=No, 3=NoExit, 4=Continue): 4 Threshold (Output): 2 Maximum Block Size (Block): 1 Output Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test4.jpg Same Method (OutputPath, 2=No, 3=NoExit, 4=Continue): 4 === MSG: MSG === 1. Waktu Eksekusi : 570.38 ms 2. Ukuran Setelah : 1748832 bytes 3. Ukuran Sebelum : 184320 bytes 4. Perbandingan Rasio : 92.2% 5. Hasil Gambar : 2 6. Jumlah Gambar : 1 7. Gambar Output : C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test4.jpg </pre>

5. Testing 5

Gambar Sebelum	Gambar Sesudah	CLI
		<pre> Input Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test5.jpg Error Method (InputPath, 2=No, 3=NoExit, 4=Continue): 4 Threshold (Output): 2 Maximum Block Size (Block): 1 Output Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test5.jpg Same Method (OutputPath, 2=No, 3=NoExit, 4=Continue): 4 === MSG: MSG === 1. Waktu Eksekusi : 1442.38 ms 2. Ukuran Setelah : 184320 bytes 3. Ukuran Sebelum : 184320 bytes 4. Perbandingan Rasio : 100% 5. Hasil Gambar : 1 6. Jumlah Gambar : 1 7. Gambar Output : C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test5.jpg </pre>

6. Testing 6

Gambar Sebelum	Gambar Sesudah	CLI
		<pre> Input Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test6.jpg Error Method (InputPath, 2=No, 3=NoExit, 4=Continue): 4 Threshold (Output): 2 Maximum Block Size (Block): 1 Output Path: C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test6.jpg Same Method (OutputPath, 2=No, 3=NoExit, 4=Continue): 4 === MSG: MSG === 1. Waktu Eksekusi : 2191.41 ms 2. Ukuran Setelah : 184320 bytes 3. Ukuran Sebelum : 184320 bytes 4. Perbandingan Rasio : 100% 5. Hasil Gambar : 1 6. Jumlah Gambar : 1 7. Gambar Output : C:\Users\DEBA-LAPTOP\Documents\Tugas12_2022\12\test\test6.jpg </pre>

7. Testing 7

Gambar Sebelum	Gambar Sesudah	CLI
----------------	----------------	-----

BAB IV

ANALISIS

1. **Metode perhitungan error wajib**, semuanya terdapat kalang berganda yang melakukan iterasi sebanyak $w \times h$ pixel sehingga kompleksitasnya adalah $O(wh)$. Untuk memudahkan perhitungan, diasumsikan panjang dan lebar block selalu sama sehingga kompleksitas waktunya menjadi $O(n^2)$.
2. **Metode pencarian nilai rata-rata warna** juga menggunakan kalang berganda yang mengunjungi setiap pixel pada block sehingga kompleksitasnya adalah $O(wh)$. Namun untuk memudahkan perhitungan, diasumsikan panjang dan lebar block selalu sama sehingga kompleksitas waktunya menjadi $O(n^2)$.
3. Algoritma Divide and Conquer Metode Quadtree dapat dianalisis kompleksitasnya yakni; Pada setiap rekursi, selalu memanggil **fungsi perhitungan error** dan **fungsi perhitungan nilai rata-rata warna** kemudian dilakukan pembagian area menjadi empat

Sehingga:

$$T(w, h) = \begin{cases} 2wh, & \text{jika } (w \cdot h < \text{minBlockSize}) \text{ atau } (\text{error} < \text{threshold}) \text{ atau } (\text{sub-blok} < \text{minBlockSize}) \\ 4T(w/2, h/2) + 2wh, & \text{lainnya} \end{cases}$$

Apabila dihitung maka:

$$T(w, h) = 4T(w/2, h/2) + 2wh$$
$$= 4(4T(w/4, h/4) + 2 \frac{wh}{4}) + 2wh$$
$$= 4^k T(w/2^k, h/2^k) + kwh$$

misal $w = 2^k, h = 2^k \rightarrow k = 2 \log w = 2 \log h$
maka, $whT(1, 1) + wh \log(\min(w, h))$ min supaya diambil waktu terbaik serta rekursi juga berhenti ketika salah satu dari w atau h habis dibagi sehingga tidak memenuhi minimum block size
$$\therefore T(w, h) = O(wh \log(\min(w, h)))$$

Apabila diasumsikan $w = h$ maka:

$$T(n) = 4T(n/2) + 2n^2$$

dengan teorema master
$$a = 4 \quad b = 2 \quad c = 2 \quad d = 2 \rightarrow a > b^d \rightarrow O(n^2 \log n)$$

$$4 = 2^2$$

Perhitungan tersebut merupakan kasus terburuk (worst case) sebab diasumsikan bahwa pada setiap pembagian sub-block mengalami rekursi kembali sampai menghasilkan sebuah pohon yang seimbang dimana perbedaan tinggi masing-masing upa-pohon tidak terlalu jauh.

Lain halnya dengan kasus terbaik (best case) ataupun kasus rata-rata (average case) yang memungkinkan tidak semua sub-block mengalami rekursi kembali akibat tidak memenuhi persyaratan, maka tentu kompleksitasnya bisa lebih rendah.

Namun untuk menyederhanakan laporan ini, kedua kasus tersebut tidak dihitung sehingga dianggap algoritma selalu menghasilkan kasus terburuk. Hal tersebut dapat dipandang positif, sebab worst case menunjukkan batas atas dari performa algoritma sehingga dapat memastikan algoritma tetap efisien bahkan ketika dihadapkan pada input paling buruk.

BAB V

KESIMPULAN

Proyek ini telah mengimplementasikan metode kompresi gambar berbasis *quadtree* dengan baik. Pendekatan yang digunakan memanfaatkan pembagian gambar menjadi blok-blok kecil berdasarkan tingkat keseragaman warna, sehingga memungkinkan penyederhanaan data tanpa mengorbankan terlalu banyak kualitas visual. Hasil kompresi menunjukkan pengurangan ukuran file yang cukup signifikan dengan tetap mempertahankan struktur dan warna utama gambar.

Selain itu, Proses input, pemrosesan, hingga penyimpanan gambar kompresi dapat dilakukan secara otomatis dan terstruktur. Dengan parameter-parameter yang dapat disesuaikan, metode ini cukup fleksibel untuk digunakan pada berbagai jenis gambar dan kebutuhan kompresi.

SARAN

Untuk kedepannya, sistem dapat dikembangkan lebih lanjut agar lebih fleksibel dan efisien. Salah adalah memberikan tampilan visual atau laporan yang lebih informatif agar pengguna dapat memahami hasil kompresi dengan lebih mudah.

CHECKLIST

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	✓	

PRANALA GITHUB

https://github.com/iqbalhaidr/Tucil2_13523111