

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



Oleh:

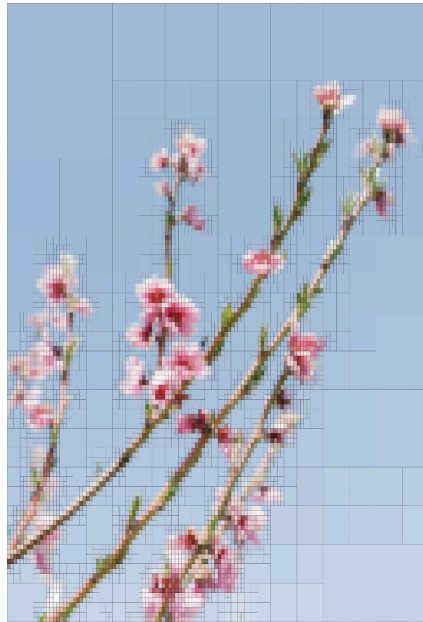
13523128 - Andi Farhan Hidayat

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

Daftar Isi

1. Topik: Kompresi Dengan Quadtree.....	3
2. Dasar Teori.....	4
2.1. Quadtree.....	4
2.2. Algoritma Divide and Conquer.....	4
2.3. Error Measurement Methods.....	4
3. Solusi (Implementasi).....	6
4.1. Test Case Variance.....	12

1. Topik: Kompresi Dengan Quadtree



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.

2. Dasar Teori

2.1. Quadtree

Quadtree merupakan struktur data hierarkis berbasis pohon yang digunakan untuk merepresentasikan partisi spasial dua dimensi secara efisien. Dalam konteks pemrosesan citra, Quadtree membagi gambar menjadi blok-blok persegi yang lebih kecil berdasarkan kriteria tertentu, seperti tingkat keseragaman warna atau error kompresi. Jika suatu blok memenuhi syarat homogenitas, maka tidak dibagi lebih lanjut dan disimpan sebagai simpul daun (leaf). Pendekatan ini sangat berguna dalam kompresi gambar, karena mampu mengurangi data tanpa mengorbankan banyak detail visual, terutama pada area gambar yang relatif seragam.

2.2. Algoritma Divide and Conquer

Divide and Conquer adalah paradigma algoritma yang menyelesaikan masalah dengan membaginya menjadi submasalah yang lebih kecil, menyelesaikan masing-masing sub masalah secara rekursif, kemudian menggabungkan hasilnya untuk membentuk solusi akhir. Strategi ini terdiri dari tiga tahap utama:

- Divide – Memecah masalah menjadi beberapa bagian kecil yang sejenis.
- Conquer – Menyelesaikan setiap bagian secara rekursif.
- Combine – Menggabungkan solusi dari bagian-bagian kecil menjadi solusi keseluruhan.

Dalam konteks Quadtree, pendekatan Divide and Conquer diterapkan dengan membagi gambar menjadi empat kuadran secara rekursif. Jika suatu kuadran masih terlalu kompleks (berdasarkan error atau ukuran tertentu), maka proses pembagian dilanjutkan hingga kuadran tersebut cukup homogen atau mencapai batas minimum. Hal ini membuat Quadtree menjadi implementasi dari prinsip *Divide and Conquer* dalam pemrosesan citra.

2.3. Error Measurement Methods

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

Tabel 1. Metode Pengukuran Error

Metode	Formula
--------	---------

<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$
	$\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok
Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c
<u>Entropy</u>	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
	$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok

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

3. Solusi (Implementasi)

3.1. Struktur Data

3.1.1. Pixel

```
struct Pixel {
    unsigned char r, g, b;
};
```

Pixel terdiri atas unsigned char r, g, b masing-masing bernilai 0 hingga 255.

3.1.2. ImageMatrix

```
using ImageMatrix = vector<vector<Pixel>>;
```

Matriks dengan tiap cell bertipe Pixel.

3.1.3. Image

```
class Image {
public:
    ImageMatrix data;
    int width, height;

    bool load(const string &path);
    bool save(const string &path);

    ImageMatrix getBlock(int x, int y, int blockWidth, int blockHeight);
    void fillBlock(int x, int y, int blockWidth, int blockHeight, Pixel color);
};
```

Berisi data keseluruhan terkait pemrosesan image, denagn atribut data, lebar, dan tingga. Serta fungsi load, save, getBlock, dan fillBlock untuk memudahkan pada pemrosesan compress image dengan quadtree.

3.1.4. AverageColor

```
struct AverageColor {  
    double r, g, b;  
};
```

AverageColor dengan data rata-rata nilai rgb untuk memudahkan perhitungan error.

3.1.5. QuadNode

```
struct QuadNode {  
    int x, y, width, height;  
    Pixel color;  
    bool isLeaf;  
    QuadNode *children[4];  
  
    QuadNode(int x, int y, int width, int height);  
    ~QuadNode();  
};
```

Berisi data posisi x dan y, lebar, serta tinggi node sebagai pecahan dari gambar. Terdapat atribut color untuk menentukan warna node ketika menjadi leaf yang ditandai dengan boolean isLeaf. QuadNode sebagai bagian dari QuadTree dapat memiliki 4 child.

3.1.6. QuadTree

```
class QuadTree {
private:
    QuadNode *root;
    Image *image;
    ErrorMethod method;
    double threshold;
    int minBlockSize;

    QuadNode *compress(int x, int y, int width, int height);

public:
    QuadTree(Image *img, ErrorMethod method, double threshold,
            int minBlockSize);
    ~QuadTree();

    void compress();
    void reconstructImage();
    int getTreeDepth();
    int getNodeCount();

    QuadNode *getRoot() const;
};
```

Berisi atribut pointer ke node root, data image, method, threshold, dan minimum block size untuk pemrosesan compress image dengan quadtree. Terdapat pula fungsi untuk mendapatkan kedalaman tree, menghitung jumlah node, me-compress gambar, dan merekonstruksi gambar.

3.2. Image Processing dengan FreeImage

3.3. Error Measurement Method

3.3.1. Variance

```
case VARIANCE: {
    for (const auto &row : block)
        for (const auto &p : row)
            err += pow(p.r - avg.r, 2) + pow(p.g - avg.g, 2) +
                pow(p.b - avg.b, 2);
    return err / n;
}
```

Variance adalah metode pengukuran error yang menghitung seberapa besar penyebaran nilai warna dalam suatu blok gambar terhadap nilai rata-ratanya.

Prosesnya dimulai dengan menghitung nilai rata-rata masing-masing *channel* warna (merah, hijau, biru), lalu menghitung selisih kuadrat antara setiap piksel dengan nilai rata-rata tersebut. Nilai ragam diperoleh dari rata-rata seluruh hasil kuadrat selisih, semakin tinggi nilainya maka semakin beragam pula nilai warna dalam blok tersebut.

3.3.2. Mean Absolute Deviation (MAD)

```
case MEAN_ABSOLUTE_DEVIATION: {
    for (const auto &row : block)
        for (const auto &p : row)
            err += abs(p.r - avg.r) + abs(p.g - avg.g) + abs(p.b -
                avg.b);
    return err / n;
}
```

Mean Absolute Deviation (MAD) mengukur deviasi nilai piksel terhadap rata-ratanya secara absolut. Setelah nilai rata-rata warna tiap *channel* diperoleh, setiap piksel dibandingkan dengan rata-rata tersebut lalu dihitung selisih absolutnya. Rata-rata dari semua deviasi absolut ini menjadi nilai MAD. Metode ini memberikan ukuran keseragaman warna dalam blok tanpa memperhitungkan arah deviasi, sehingga lebih tahan terhadap outlier daripada *variance*.

3.3.3. Max Pixel Difference

```
case MAX_PIXEL_DIFFERENCE: {
    int minR = 255, minG = 255, minB = 255;
    int maxR = 0, maxG = 0, maxB = 0;
    for (const auto &row : block)
        for (const auto &p : row) {
            if (p.r > maxR) maxR = p.r;
            if (p.r < minR) minR = p.r;

            if (p.g < minG) minG = p.g;
            if (p.g > maxG) maxG = p.g;

            if (p.b < minB) minB = p.b;
            if (p.b > maxB) maxB = p.b;
        }
    return (maxR - minR + maxG - minG + maxB - minB) / 3.0;
}
```

Max Pixel Difference digunakan untuk mengukur rentang intensitas warna dalam blok. Dalam metode ini, setiap nilai warna piksel diperiksa untuk menentukan nilai maksimum dan minimum pada masing-masing *channel*. Selisih antara nilai maksimum dan minimum dihitung untuk setiap *channel*, kemudian dirata-rata. Nilai ini merepresentasikan seberapa besar kontras warna yang ada dalam satu blok.

3.3.4. Entropy

```
case ENTROPY: {
    std::vector<int> histR(256, 0), histG(256, 0), histB(256, 0);

    for (const auto &row : block) {
        for (const auto &p : row) {
            histR[p.r]++;
            histG[p.g]++;
            histB[p.b]++;
        }
    }

    auto computeChannelEntropy =
    [&](const std::vector<int> &hist) -> double {
        double entropy = 0.0;
        for (int count : hist) {
            if (count > 0) {
                double p = static_cast<double>(count) / n;
                entropy -= p * std::log2(p);
            }
        }
        return entropy;
    };

    double entropyR = computeChannelEntropy(histR);
    double entropyG = computeChannelEntropy(histG);
    double entropyB = computeChannelEntropy(histB);

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

Entropy merupakan ukuran kompleksitas atau keacakan distribusi warna dalam blok gambar. Dengan membangun histogram frekuensi kemunculan tiap nilai warna (0–255) untuk setiap *channel*, probabilitas kemunculan dihitung dan digunakan dalam rumus entropi Shannon. Hasil dari ketiga *channel* dirata-rata untuk mendapatkan nilai entropi blok. Nilai entropi yang tinggi menunjukkan distribusi warna yang lebih acak dan kompleks.

3.3.5. Structural Similarity Index (SSIM)

```
case SSIM: {
    if (!reconstructed)
        throw invalid_argument("SSIM requires a reconstructed block.");

    const double C1 = 6.5025; // (0.01 * 255)^2
    const double C2 = 58.5225; // (0.03 * 255)^2

    auto computeLuminance = [](const Pixel &p) -> double {
        return 0.299 * p.r + 0.587 * p.g + 0.114 * p.b;
    };

    double mu_x = 0.0, mu_y = 0.0;
    for (size_t i = 0; i < block.size(); ++i)
        for (size_t j = 0; j < block[0].size(); ++j) {
            mu_x += computeLuminance(block[i][j]);
            mu_y += computeLuminance((*reconstructed)[i][j]);
        }
    mu_x /= n;
    mu_y /= n;

    double sigma_x = 0.0, sigma_y = 0.0, sigma_xy = 0.0;
    for (size_t i = 0; i < block.size(); ++i)
        for (size_t j = 0; j < block[0].size(); ++j) {
            double x = computeLuminance(block[i][j]);
            double y = computeLuminance((*reconstructed)[i][j]);
            sigma_x += (x - mu_x) * (x - mu_x);
            sigma_y += (y - mu_y) * (y - mu_y);
            sigma_xy += (x - mu_x) * (y - mu_y);
        }

    sigma_x /= (n - 1);
    sigma_y /= (n - 1);
    sigma_xy /= (n - 1);

    double nume = (2 * mu_x * mu_y + C1) * (2 * sigma_xy + C2);
    double deno =
        (mu_x * mu_x + mu_y * mu_y + C1) * (sigma_x + sigma_y + C2);
    double ssim = nume / deno;

    return 1.0 - ssim;
}
```

Structural Similarity Index (SSIM) adalah metode canggih yang membandingkan dua blok gambar, yaitu blok asli dan blok hasil rekonstruksi, berdasarkan kemiripan strukturnya. SSIM menghitung luminansi (terang-gelap), kontras, dan struktur antara dua blok menggunakan parameter statistik seperti rata-rata, variansi, dan kovarian. Nilai SSIM berkisar dari 0 (tidak mirip) hingga 1 (sangat mirip) dan dalam konteks pengukuran error ini, nilai yang digunakan adalah 1 dikurangi SSIM. Hal tersebut lebih cocok secara visual karena mempertimbangkan persepsi manusia terhadap kualitas gambar.

3.4. Quadtree

Algoritma Divide and Conquer dalam kompresi gambar berbasis Quadtree pada kode program ini bekerja dengan cara membagi (*divide*) gambar menjadi blok-blok yang lebih kecil secara rekursif, kemudian menyelesaikan (*conquer*) tiap bagian dengan menganalisis apakah blok cukup sesuai untuk tidak perlu dibagi lagi berdasarkan data *threshold* dan *minimum block size*. Jika tidak sesuai, maka blok dibagi lagi ke dalam empat kuadran (sub-blok) dan proses diulang hingga kondisi pemberhentian (*base case*) tercapai.

Berikut langkah-langkah algoritmanya secara konseptual:

1. Inisialisasi

Algoritma dimulai dari satu blok gambar penuh (dari koordinat (0,0) hingga ke ujung kanan bawah gambar). Fungsi `compress()` memulai proses dengan memanggil fungsi rekursif `compress(x, y, width, height)`.

```
void QuadTree::compress() {  
    root = compress(0, 0, image->width, image->height);  
}
```

Gambar 3.4.1. Gambar inisialisasi `compress` pada koordinat (0,0)

2. Divide (Pembagian Blok)

Untuk setiap blok yang sedang dianalisis, algoritma menghitung error dengan method yang ada, yaitu: Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy, dan Structural Similarity Index (SSIM) untuk mengetahui tingkat homogenitas warna dalam blok tersebut.

Jika error melebihi ambang batas (*threshold*) dan ukuran blok masih lebih besar dari batas minimum (*minBlockSize*), maka blok dibagi menjadi empat sub-blok (kuadran) yaitu:

- Kiri atas
- Kanan atas
- Kiri bawah
- Kanan bawah

```
node->children[0] = compress(x, y, halfW, halfH);  
node->children[1] = compress(x + halfW, y, width - halfW, halfH);  
node->children[2] = compress(x, y + halfH, halfW, height - halfH);  
node->children[3] =  
    compress(x + halfW, y + halfH, width - halfW, height - halfH);
```

Gambar 3.4.2. Pembagian blok menjadi empat sub-blok

Masing-masing sub-blok tersebut akan dianalisis ulang secara rekursif.

Conquer (Penyelesaian Blok)

Jika error blok berada di bawah threshold atau ukuran blok sudah terlalu kecil untuk dibagi lebih lanjut, maka blok dianggap cukup homogen dan tidak perlu dipecah lagi. Blok ini ditandai sebagai leaf node dan disimpan bersama dengan warna rata-rata dari piksel-piksel di blok tersebut.

```
double error = (method == SSIM)
    ? computeError(block, method, &reconstructed)
    : computeError(block, method, nullptr);

if (error <= threshold || (width * height) <= minBlockSize) {
    auto node = new QuadNode(x, y, width, height);
    node->isLeaf = true;
    node->color = averageColor(block);
    return node;
}
```

Gambar 3.4.3. Algoritma umum penyelesaian (*conquer*) blok

Gabungan Solusi (*Combine*)

Struktur Quadtree dibentuk secara bertahap dari hasil rekursi. Setiap node menyimpan referensi keempat *child*-nya (jika bukan leaf), sehingga membentuk hierarki pohon dari gambar asli. Informasi ini kemudian dapat digunakan untuk merekonstruksi gambar atau menghitung statistik seperti kedalaman dan jumlah node.

```
void fill(Image *image, QuadNode *node) {
    if (!node) return;
    if (node->isLeaf) {
        image->fillBlock(node->x, node->y, node->width, node->height,
            node->color);
    } else {
        for (auto &child : node->children)
            fill(image, child);
    }
}

void QuadTree::reconstructImage() { fill(image, root); }
```

Gambar 3.4.4. Rekonstruksi gambar dengan rekursif mengisi gambar berdasarkan *child*

Ketika semua cabang rekursi selesai (seluruh blok telah dianalisis dan tidak bisa dibagi lagi), maka kompresi gambar selesai. Hasil akhirnya adalah struktur pohon Quadtree yang menyimpan blok-blok homogen dari gambar.

Dengan demikian, pendekatan Divide and Conquer di Quadtree ini melibatkan:

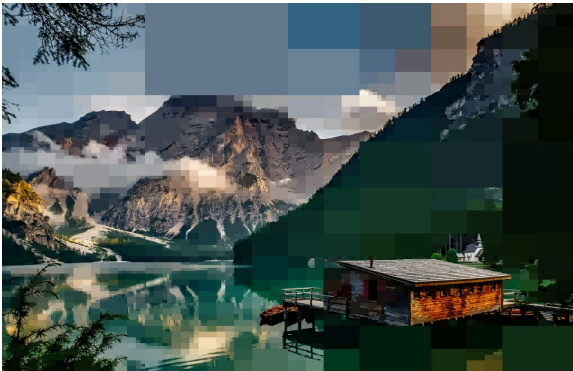
- *Divide*: membagi blok menjadi empat bagian jika tidak homogen.
- *Conquer*: menghentikan pembagian jika blok cukup homogen atau terlalu kecil.
- *Combine*: membangun struktur pohon dari hasil blok-blok yang telah diproses.

4. Pengujian

Keterangan urutan input:

- Absolute input image path
- Method
- Threshold
- Minimum block size
- Target Compression
- Absolute result image path
- Absolute result gif path

4.1. Test Case Variance


Input	Output
<ul style="list-style-type: none"> • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg • 1 • 2000 • 10 • 0 • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result1.jpg • - 	 <p>Image successfully compressed! [OUTPUT] execution time: 1752ms [OUTPUT] image size before: 543764 bytes [OUTPUT] image size after: 323425 bytes [OUTPUT] compression percentage: 40.5211% [OUTPUT] tree depth: 10 [OUTPUT] node count: 145145 [OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result1.jpg</p>

Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 1
[INPUT] Threshold: 2000
[INPUT] Minimum block size: 10
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result1.jpg
[INPUT] Absolute result gif path: -

Loading...
Image successfully compressed!
[OUTPUT] execution time: 1752ms
[OUTPUT] image size before: 543764 bytes
[OUTPUT] image size after: 323425 bytes
[OUTPUT] compression percentage: 40.5211%
[OUTPUT] tree depth: 10
[OUTPUT] node count: 145145
[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result1.jpg
```

4.2. Test Case Mean Absolute Deviation (MAD)

Input	Output
<ul style="list-style-type: none"> • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg • 2 • 20 • 10 • 0 • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result2.jpg • - 	 <p>Image successfully compressed!</p> <p>[OUTPUT] execution time: 1845ms</p> <p>[OUTPUT] image size before: 543764 bytes</p> <p>[OUTPUT] image size after: 500133 bytes</p> <p>[OUTPUT] compression percentage: 8.02389%</p> <p>[OUTPUT] tree depth: 10</p> <p>[OUTPUT] node count: 534221</p> <p>[OUTPUT] compressed image path:</p>


	/mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result2.jpg
--	--------------------------------------------------------------

Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 2
[INPUT] Threshold: 20
[INPUT] Minimum block size: 10
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result2.jpg
[INPUT] Absolute result gif path: -

Loading...
Image successfully compressed!
[OUTPUT] execution time: 1845ms
[OUTPUT] image size before: 543764 bytes
[OUTPUT] image size after: 500133 bytes
[OUTPUT] compression percentage: 8.02389%
[OUTPUT] tree depth: 10
[OUTPUT] node count: 534221
[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result2.jpg
```

4.3. Test Case Max Pixel Difference

Input	Output
<ul style="list-style-type: none"> • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg • 3 • 20 • 10 • 0 • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result3.jpg • - 	 <p>Image successfully compressed!</p> <p>[OUTPUT] execution time: 2248ms</p> <p>[OUTPUT] image size before: 543764 bytes</p> <p>[OUTPUT] image size after: 523737 bytes</p> <p>[OUTPUT] compression percentage:</p>


	3.68303% [OUTPUT] tree depth: 10 [OUTPUT] node count: 642761 [OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result3.jpg
--	-----------------------------------------------------------------------------------------------------------------------------------------------------------------------

Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 3
[INPUT] Threshold: 20
[INPUT] Minimum block size: 10
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result3.jpg
[INPUT] Absolute result gif path: -

Loading...
Image successfully compressed!
[OUTPUT] execution time: 2248ms
[OUTPUT] image size before: 543764 bytes
[OUTPUT] image size after: 523737 bytes
[OUTPUT] compression percentage: 3.68303%
[OUTPUT] tree depth: 10
[OUTPUT] node count: 642761
[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result3.jpg
```

4.4. Test Case Entropy

Input	Output
<ul style="list-style-type: none"> • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg • 4 • 3 • 10 • 0 • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result4.jpg • - 	 <p>Image successfully compressed! [OUTPUT] execution time: 4716ms</p>

	[OUTPUT] image size before: 543764 bytes [OUTPUT] image size after: 512521 bytes [OUTPUT] compression percentage: 5.74569% [OUTPUT] tree depth: 10 [OUTPUT] node count: 629229 [OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result4.jpg
--	--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------


Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 4
[INPUT] Threshold: 3
[INPUT] Minimum block size: 10
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result4.jpg
[INPUT] Absolute result gif path: -

Loading...
Image successfully compressed!
[OUTPUT] execution time: 4716ms
[OUTPUT] image size before: 543764 bytes
[OUTPUT] image size after: 512521 bytes
[OUTPUT] compression percentage: 5.74569%
[OUTPUT] tree depth: 10
[OUTPUT] node count: 629229
[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result4.jpg
```

4.5. Test Case Structural Similarity Index (SSIM)

Input	Output
-------	--------

<ul style="list-style-type: none"> • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg • 5 • 0.8 • 10 • 0 • /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result5.jpg • - 	 <p>Image successfully compressed!</p> <p>[OUTPUT] execution time: 2507ms</p> <p>[OUTPUT] image size before: 543764 bytes</p> <p>[OUTPUT] image size after: 463349 bytes</p> <p>[OUTPUT] compression percentage: 14.7886%</p> <p>[OUTPUT] tree depth: 11</p> <p>[OUTPUT] node count: 434153</p> <p>[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result5.jpg</p>
------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------

Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 5
[INPUT] Threshold: 0.8
[INPUT] Minimum block size: 4
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result5.jpg
[INPUT] Absolute result gif path: -

Loading...
Image successfully compressed!
[OUTPUT] execution time: 2507ms
[OUTPUT] image size before: 543764 bytes
[OUTPUT] image size after: 463349 bytes
[OUTPUT] compression percentage: 14.7886%
[OUTPUT] tree depth: 11
[OUTPUT] node count: 434153
[OUTPUT] compressed image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/result5.jpg
```

4.6. Test Case Input Metode Tidak Tepat

Input	Output
<ul style="list-style-type: none">• /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg• 6	[ERROR] Invalid method: must be between 1 and 5.

Screenshot CLI:

```
[INPUT] Absolute image path: /mnt/c/ITB/Semester_4/Stima/Tucil2_13523128/test/dream.jpg
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 6
[ERROR] Invalid method: must be between 1 and 5.
```

4.7. Test Case Input File Path Tidak Tepat

Input	Output
<ul style="list-style-type: none">• 1231231231• 1• 10• 10• 0• 12312312312• -	Failed to load image.

Screenshot CLI:

```

[INPUT] Absolute image path: 1231231231
Error Measurement Methods:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM)
[INPUT] Chosen method (number only): 1
[INPUT] Threshold: 10
[INPUT] Minimum block size: 10
[INPUT] Target compression percentage in floating number (1.0 = 100%) (0 to disable): 0
[INPUT] Absolute result image path: 12312312312
[INPUT] Absolute result gif path: -

Loading...
Failed to load image.

```

5. Analisis

5.1. Efektivitas

Algoritma Quadtree secara efektif membagi wilayah citra berdasarkan homogenitas warna. Area gambar yang memiliki perbedaan warna kecil (misalnya latar belakang polos) akan langsung diubah menjadi leaf node tanpa perlu dibagi lebih lanjut, sementara bagian yang memiliki variasi warna tinggi (misalnya detail objek atau tepi) akan dipecah menjadi blok-blok lebih kecil. Hal ini memungkinkan kompresi yang adaptif tergantung pada bentuk dan komposisi gambar. Pada test case yang telah saya lakukan, terlihat bahwa terdapat persentase kompresi yang kecil. Hal tersebut karena gambar yang digunakan untuk pengujian terlalu rinci dan komposisinya tidak terlalu polos kecuali pada bagian langit. Hal ini membuat quadtree terus membagi dirinya hingga mencapai minimum block size sehingga kompresi yang dilakukan tidak begitu besar. Hal tersebut juga dibuktikan dengan jumlah *node* yang besar dan *depth tree* yang dalam.

5.2. Kompleksitas Waktu Algoritma

Dalam kasus terburuk, ketika seluruh blok tidak pernah memenuhi syarat sebagai leaf dan terus dibagi hingga mencapai ukuran minimum blok (`minBlockSize`), kompleksitas waktu menjadi $O(n^2)$, dengan n adalah panjang sisi citra (dengan

asumsi citra persegi). Hal ini terjadi karena setiap pembagian menghasilkan 4 sub-blok, dan proses dilakukan secara rekursif.

$$T(n) = 1 + 4 + 4^2 + \dots + 4^k$$

Dengan $k = \log_2(n)$.

Lalu jika dijumlahkan keseluruhannya maka akan didapatkan

$$T(n) \Rightarrow (4^{(\log_2(n)+1)} - 1) / 3 = O(n^2)$$

Namun dalam kasus rata-rata (pada citra nyata), kompleksitas dapat lebih rendah karena keberadaan area homogen yang membuatnya tidak perlu dibagi.

5.3. Kompleksitas Ruang dan Struktur Pohon

Struktur pohon Quadtree membutuhkan memori tambahan untuk menyimpan node dan anak-anaknya. Jumlah total node pada pohon bergantung pada kompleksitas visual citra dan nilai ambang batas error (*threshold*). Semakin tinggi *threshold*, semakin sedikit *node* yang terbentuk karena lebih banyak blok dianggap homogen. Namun semakin rendah *threshold*, semakin dalam pohon dan lebih banyak node dibentuk.

5.4. Trade-off antara Kualitas dan Ukuran Kompresi

Nilai ambang batas error (*threshold*) dan ukuran minimum blok (*minBlockSize*) sangat mempengaruhi hasil akhir. *Threshold* yang tinggi meningkatkan rasio kompresi tetapi menurunkan kualitas visual karena lebih banyak blok disederhanakan. Sebaliknya, *threshold* rendah menjaga detail gambar, tetapi menurunkan efektivitas kompresi karena pohon menjadi lebih kompleks dan blok lebih kecil.

5.5. Skabilitas dan Kinerja pada Resolusi Tinggi

Pada gambar resolusi tinggi, performa algoritma masih cukup baik karena bersifat lokal, yaitu hanya area dengan kompleksitas tinggi yang dianalisis lebih dalam. Hal ini membuat algoritma Quadtree cocok untuk gambar besar yang memiliki banyak

area homogen. Namun demikian, performa tetap bergantung pada implementasi fungsi penghitungan error.

6. Lampiran

6.1. Tabel Poin

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	v	
2. Program berhasil dijalankan	v	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	v	
4. Mengimplementasi seluruh metode perhitungan error wajib	v	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan		v
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error	v	
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar		v
8. Program dan laporan dibuat (kelompok) sendiri	v	

6.2. Pranala Repository GitHub Kode Program

https://github.com/andi-frame/Tucil2_13523128