

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



**Oleh:**  
**Muhammad Fithra Rizki - 13523049**  
**Farrel Athalla Putra - 13523118**

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

## DAFTAR ISI

<b>DAFTAR ISI</b>	<b>2</b>
<b>BAB I</b>	
<b>DESKRIPSI MASALAH DAN ALGORITMA</b>	<b>4</b>
1.1 Algoritma Divide and Conquer	4
1.2 Kompresi Gambar Dengan Metode Quadtree	5
1.3 Algoritma Kompresi Gambar dengan Metode Quadtree Melalui Pendekatan Divide and Conquer	6
<b>BAB II</b>	
<b>IMPLEMENTASI ALGORITMA DALAM BAHASA C++</b>	<b>9</b>
2.1 File ErrorCalculator.cpp	9
2.2 File IO.cpp	10
2.3 File QuadTree.cpp	10
2.4 Reconstruct.cpp	12
2.5 File main.cpp	13
<b>BAB III</b>	
<b>SOURCE CODE</b>	<b>14</b>
2.1 IO.cpp	14
2.2 ErrorCalculator.cpp	16
2.3 QuadTree.cpp	18
2.4 Reconstruct.cpp	20
2.5 main.cpp	23
<b>BAB IV</b>	
<b>TESTING</b>	<b>27</b>
3.1 Metode Variance	27
3.2 Metode Mean Absolute Deviation	29
3.3 Metode Max Pixel Difference	31
3.4 Metode Entropy	33
3.5 Metode Structure Similarity Index	35
3.6 Black and White	37
3.7 Target Kompresi	38
<b>BAB V</b>	
<b>IMPLEMENTASI BONUS</b>	<b>41</b>
5.1 Bonus Target Persentase Kompresi	41
5.2 Bonus Structure Similarity Index (SSIM)	42
5.3 Bonus GIF	42
<b>BAB VI</b>	
<b>ANALISIS PROGRAM</b>	<b>44</b>
<b>BAB VII</b>	
<b>KESIMPULAN</b>	<b>46</b>
<b>LAMPIRAN</b>	<b>47</b>
<b>DAFTAR PUSTAKA</b>	<b>48</b>

## **BAB I**

### **DESKRIPSI MASALAH DAN ALGORITMA**

#### **1.1 Algoritma *Divide and Conquer***

Divide and Conquer adalah sebuah strategi pemecahan masalah yang dimana melibatkan tiga proses utama, yakni divide, conquer, dan combine. Proses divide berisi proses pembagian masalah menjadi upa-persoalan yang serupa dengan persoalan aslinya, tetapi dalam ukuran-ukuran yang sudah terbagi dan lebih kecil. Secara ideal, upa-persoalan upa-persoalan ini memiliki ukuran yang hampir seragam. Lalu, proses conquer merupakan proses penyelesaian masing-masing upa-persoalan yang berukuran kecil atau terus-menerus membagi secara rekursif upa-persoalan yang masih berukuran besar. Terakhir, proses combine berarti menggabungkan seluruh solusi-solusi yang didapat dari masing-masing upa-persoalan yang berukuran kecil sehingga dapat membentuk solusi utama untuk persoalan utama. Metode divide and conquer sangat berguna dalam menyelesaikan permasalahan-permasalahan kompleks karena membagi permasalahan menjadi upa-permasalahan yang merupakan bagian-bagian yang mudah diatasi dan menggabungkan solusi-solusi tersebut diakhir untuk mendapat solusi akhir dari permasalahan utama.

Metode divide and conquer seringkali menjadi metode yang lebih efisien dari metode lainnya, hal inilah yang menjadi keunggulan dari strategi ini. Efisiensi dapat dilihat dari kompleksitas penyelesaian suatu masalah. Contoh persoalan yang membuktikan bahwa strategi ini menjadi strategi yang efisien adalah menentukan nilai maksimum dan minimum dari suatu larik. Dari permasalahan tersebut, jika digunakan strategi *brute force* untuk penyelesaian masalahnya, didapat kompleksitas waktunya sebesar  $T(n) = 2n - 2$ . Sedangkan, jika permasalahan tersebut diselesaikan dengan strategi divide and conquer, didapat kompleksitas waktunya  $T(n) = 3n/2 - 2$ . Dapat dilihat bahwa untuk  $n \geq 2$ ,  $3n/2 - 2$  memiliki nilai yang lebih kecil dari  $2n-2$ . Hal ini menunjukan bahwa strategi divide and conquer lebih sangkil dalam memecahkan masalah pencarian nilai maksimum dan minimum. Hal lain yang membuat strategi ini unggul adalah memori cache yang digunakan pada strategi ini cenderung efisien, sehingga sumber daya memori yang ada digunakan secara optimal.

## **1.2 Kompresi Gambar Dengan Metode Quadtree**

Kompresi gambar merupakan proses mengurangi ukuran data dari representasi citra digital tanpa mengorbankan terlalu banyak kualitas visual dari gambar tersebut. Salah satu teknik kompresi yang efisien dan banyak digunakan dalam pengolahan citra adalah metode Quadtree.

Quadtree adalah struktur data pohon hierarkis yang dirancang khusus untuk representasi spasial dua dimensi. Istilah "Quadtree" berasal dari fakta bahwa setiap simpul internal pada pohon memiliki tepat empat anak yang merepresentasikan empat bagian dari area dua dimensi: kuadran kiri atas, kanan atas, kiri bawah, dan kanan bawah. Konsep ini pertama kali diperkenalkan oleh Finkel dan Bentley (1974) untuk kebutuhan partisi spasial pada peta dan citra digital, dan dikembangkan lebih lanjut oleh Samet (1984) dalam bidang representasi citra dan basis data spasial. Dalam konteks kompresi gambar, Quadtree digunakan untuk merepresentasikan sebuah gambar sebagai sekumpulan blok berukuran variabel berdasarkan homogenitas piksel dalam blok tersebut. Proses dimulai dengan memperlakukan keseluruhan gambar sebagai satu blok besar. Kemudian, blok ini diperiksa untuk menentukan apakah piksel-piksel di dalamnya cukup seragam (homogen). Homogenitas ini dievaluasi dengan metode-metode tertentu, seperti Variance yang mengukur seberapa besar sebaran nilai piksel dari rata-rata, Mean Absolute Deviation (MAD) yang mengukur rata-rata dari nilai selisih absolut tiap piksel terhadap rata-rata warna, Maximum Pixel Difference, mengukur selisih maksimum antara piksel tertinggi dan terendah dalam blok, Entropy mengukur kompleksitas informasi dalam blok, dan SSIM (Structural Similarity Index) yang mengukur kesamaan struktural antar blok.

Jika nilai error berada di bawah ambang batas tertentu, blok tersebut dianggap homogen dan direpresentasikan sebagai simpul daun (leaf node) dalam pohon. Sebaliknya, jika tidak homogen, blok dibagi menjadi empat bagian (kuadran), dan proses ini berulang secara rekursif pada tiap kuadran. Setiap pembagian ini akan menghasilkan node baru dalam struktur pohon, dan proses rekursi akan berhenti saat semua blok telah memenuhi kriteria homogenitas atau ukuran blok telah mencapai batas minimum yang ditentukan.

### **1.3 Algoritma Kompresi Gambar dengan Metode Quadtree Melalui Pendekatan Divide and Conquer**

Dalam menyelesaikan permasalahan kompresi gambar, dapat digunakan pendekatan QuadTree untuk membagi gambar menjadi blok-blok berdasarkan nilai error tertentu. Adapun langkah-langkah algoritma yang diterapkan dalam program dapat dijelaskan secara deskriptif sebagai berikut:

#### **1. Input Data Gambar dan Parameter Kompresi**

Program meminta pengguna untuk menginput beberapa parameter, yaitu:

- a. Path gambar input.
- b. Metode perhitungan error (Variance, MAD, Max Pixel Difference, Entropy, SSIM).
- c. Nilai ambang batas error (threshold).
- d. Ukuran minimum blok.
- e. Target persentase kompresi (opsional, 0 untuk menonaktifkan).
- f. Path output gambar dan GIF (opsional).

#### **2. Load Gambar**

Program membaca gambar menggunakan pustaka stb\_image dan mengembalikan data pixel, dimensi (lebar dan tinggi), serta jumlah kanal warna.

#### **3. Pemilihan Mode Kompresi**

- Jika `targetCompression > 0`, maka program mengaktifkan mode pencarian threshold otomatis.
- Jika `targetCompression = 0`, maka kompresi dilakukan menggunakan threshold yang ditentukan secara manual oleh pengguna.

#### **4. Mode Threshold Otomatis**

Pada mode ini, program melakukan pencarian nilai threshold terbaik menggunakan pendekatan binary search dan `minBlockSize` ditetapkan dengan nilai 1. Threshold dicoba sebanyak 10 iterasi dari rentang nilai yang disesuaikan berdasarkan metode error yang digunakan:

- Variance: 0 - 20000
- MAD & MPD: 0 - 255
- Entropy: 0.5 - 8.0

- SSIM: 0 - 1.0

Untuk setiap threshold yang dicoba, dibuat QuadTree dari image dan dilakukan kompresi gambar berdasarkan metode error yang dipilih. Setiap hasil kompresi dibandingkan dengan target kompresi dan disimpan sementara di dalam temp\_quadtree. Hasil kompresi yang paling mendekati target kompresi disimpan dalam folder sementara dengan nama best\_output dan disalin sebagai file output akhir di akhir iterasi.

## 5. Konstruksi QuadTree

Algoritma membagi gambar menjadi blok-blok secara Divide and Conquer. Proses dilakukan sebagai berikut:

1. Fungsi buildTree() menginisialisasi proses dari akar pohon (root). Root merepresentasikan seluruh gambar.
2. Untuk setiap blok, program mengambil data pixel menggunakan getBlock(). Kemudian dihitung warna rata-rata setiap blok. Setelah itu, dilakukan perhitungan error antara warna rata-rata dan piksel aktual menggunakan calculateError().
3. Jika ukuran blok sudah lebih kecil dari minBlockSize atau  $\text{error} \leq \text{threshold}$ , maka blok tidak dibagi menjadi lebih kecil dan node menjadi leaf.
4. Divide : jika blok tidak memenuhi syarat di atas, maka blok akan dibagi menjadi 4, yaitu kiri atas, kanan atas, kiri bawah, dan kanan bawah.
5. Conquer : setiap blok yang telah dibagi akan diproses kembali dari langkah 2 sampai 5. Proses ini terus dilakukan hingga seluruh blok menjadi *leaf node*.
6. Combine : setelah semua telah menjadi leaf node. Setiap gambar memiliki informasi mengenai posisi, ukuran, serta warna rata-rata blok. Gambar digabungkan kembali dengan menggambar ulang setiap blok sesuai warna rata-ratanya.

## 6. Evaluasi Error

Nilai error antar piksel dihitung menggunakan lima metode:

- Variance: Mengukur sebaran nilai warna piksel.
- MAD: Deviasi absolut rata-rata piksel terhadap mean.
- Max Pixel Difference: Selisih terbesar antar piksel.
- Entropy: Mengukur ketidakpastian distribusi warna.
- SSIM: Mengukur kesamaan struktural (semakin tinggi, semakin mirip).

## 7. Rekonstruksi dan Penyimpanan Gambar

Setelah pohon selesai dibentuk, dilakukan rekonstruksi gambar dari leaf node. Hasil disimpan dalam format bersesuaian.

## 8. Pembuatan GIF Proses Kompresi (Opsional)

Jika pengguna menginput path GIF:

- Proses traversal dilakukan berdasarkan level node.
- Setiap level hasilnya disimpan sebagai frame.
- Semua frame digabung menjadi animasi GIF menggunakan ImageMagick.

## 9. Output Statistik Kompresi

Program menampilkan: waktu eksekusi (ms), ukuran file sebelum dan sesudah kompresi, persentase rasio kompresi, kedalaman maksimum QuadTree, jumlah total simpul.

## BAB II

### IMPLEMENTASI ALGORITMA DALAM BAHASA C++

#### 2.1 File ErrorCalculator.cpp

File ini berisi implementasi berbagai metode perhitungan error (error metrics) yang digunakan dalam algoritma kompresi gambar berbasis QuadTree.

<b>Nama Fungsi</b>	<b>Deskripsi</b>
double calculateMAD(const Block& block, int width, int height)	Fungsi untuk menghitung nilai Mean Absolute Deviation (MAD) dari blok warna. MAD digunakan untuk mengukur sejauh mana piksel-piksel dalam blok menyimpang secara rata-rata dari nilai rata-ratanya.
double calculateMaxDiff(const Block& block1, const Block& block2, int width, int height)	Fungsi ini menghitung perbedaan maksimum antar dua blok gambar piksel demi piksel. Digunakan untuk melihat seberapa besar deviasi ekstrem antara blok asli dan blok hasil rata-rata.
double calculateVariance(const Block& block, int width, int height)	Menghitung variansi warna dalam blok sebagai ukuran sebaran intensitas warna. Semakin tinggi nilai variansi, semakin besar variasi warna dalam blok tersebut.
double calculateEntropy(const Block& block, int width, int height)	Menghitung entropi dari blok sebagai ukuran keacakan atau kompleksitas informasi dalam blok. Digunakan untuk mengetahui sejauh mana sebuah blok mengandung informasi yang "acak".
double calculateSSIM(const Block& block1, const Block& block2, int width, int height)	Menghitung Structural Similarity Index (SSIM) antara dua blok. SSIM lebih peka terhadap persepsi visual dan mengukur seberapa "mirip" struktur kedua blok.
double calculateError(int methodId, const Block& block1, const Block& block2, int width, int height)	Untuk memilih metode perhitungan error berdasarkan methodId dari input pengguna.

## 2.2 File IO.cpp

File ini berisi fungsi-fungsi input/output yang digunakan.

Nama Fungsi	Deskripsi
void printErrorMethodOptions()	Fungsi ini mencetak daftar metode error (error metric) yang tersedia dan dapat dipilih oleh pengguna.
InputParams getUserInput()	Fungsi utama untuk mengambil input dari pengguna berupa path, pilihan metode error, nilai threshold, ukuran minimum blok, target persentase kompresi, path output gambar, dan path output GIF.
void printOutputStats(const OutputStats& stats)	Fungsi ini menampilkan informasi statistik hasil kompresi berupa waktu eksekusi, ukuran gambar sebelum dan sesudah kompresi, persentase kompresi, kedalaman maksimum pohon, jumlah total simpul.
void printImageSavedMessage(const string& path)	Fungsi ini menampilkan pesan bahwa gambar hasil kompresi telah berhasil disimpan di path tertentu.
void printGifSavedMessage(const string& path)	Jika path untuk menyimpan GIF disediakan dan tidak kosong, maka fungsi ini akan mencetak pesan bahwa proses GIF telah berhasil disimpan.
unsigned char* load_image(const char* filename, int& width, int& height, int& channels)	Fungsi ini digunakan untuk memuat gambar dari path file yang diberikan.

## 2.3 File QuadTree.cpp

File ini mengimplementasikan struktur QuadTree dan proses kompresi gambar.

Nama Fungsi	Deskripsi
QuadTree::QuadTree(const unsigned char* img, int width, int height, int channels, int minBlockSize, double threshold, int errorMethod)	Konstruktor untuk membuat objek QuadTree.

void QuadTree::buildTree()	Fungsi utama untuk memulai proses konstruksi pohon.
std::unique_ptr<QuadTreeNode> QuadTree::build(int x, int y, int width, int height, int depth)	Fungsi utama rekursif (core Divide and Conquer). Proses dimulai dengan mengambil blok piksel pada area tertentu dari gambar menggunakan fungsi getBlock. Setelah blok diperoleh, sistem akan menghitung warna rata-rata dari seluruh piksel dalam blok tersebut melalui fungsi calculateAverageColor. Selanjutnya, tingkat homogenitas blok dievaluasi dengan mengukur nilai error menggunakan metode yang dipilih pengguna (seperti Variance, MAD, atau SSIM) melalui fungsi calculateError. Jika nilai error berada di bawah ambang batas atau ukuran blok sudah mencapai batas minimum, maka blok tersebut dianggap cukup homogen dan dijadikan sebagai leaf node yang tidak akan dibagi lagi. Sebaliknya, jika blok tidak memenuhi kriteria tersebut, maka blok akan dibagi menjadi empat kuadran (top-left, top-right, bottom-left, bottom-right) dan proses yang sama akan dilakukan secara rekursif pada masing-masing bagian tersebut hingga seluruh area gambar terproses.
std::vector<uint8_t> QuadTree::getBlock(int x, int y, int width, int height)	Mengambil dan menyalin data piksel pada blok gambar tertentu dari gambar asli berdasarkan posisi dan ukuran.
std::vector<uint8_t> QuadTree::calculateAverageColor(const std::vector<uint8_t>& block)	Menghitung warna rata-rata dari blok gambar yang diberikan, dipakai untuk pengecekan homogenitas warna serta digunakan sebagai representasi blok jika menjadi leaf node.
double QuadTree::calculateError(const std::vector<uint8_t>& block, const std::vector<uint8_t>& avgColor, int width, int height)	Menghitung nilai error untuk mengevaluasi apakah suatu blok dapat direpresentasikan oleh satu warna rata-rata.
void QuadTree::collectNodesPerDepth(std::vector<std::vector<QuadTreeNode*>>& levels)	Mengumpulkan node-node pada pohon berdasarkan kedalaman (depth).

void QuadTree::collectNodesByDepth(std::map<int, std::vector<QuadTreeNode*>>& nodesByDepth)	Mengumpulkan node-node berdasarkan kedalaman.
int QuadTree::getMaxDepth()	Mengembalikan kedalaman maksimum dari pohon QuadTree.
int QuadTree::getTotalNodes()	Mengembalikan total jumlah node (baik leaf maupun internal) dalam pohon.
const std::unique_ptr<QuadTreeNode>& QuadTree::getRoot()	Mengembalikan pointer ke akar pohon.
int QuadTree::getWidth(), getHeight(), getChannels()	Getter untuk atribut dimensi dan jumlah channel gambar.

## 2.4 Reconstruct.cpp

File ini berisi program untuk merekonstruksi gambar hasil kompresi menggunakan struktur QuadTree serta menghasilkan animasi GIF proses rekonstruksinya.

Nama Fungsi	Deskripsi
void reconstructNode(QuadTreeNode* node, unsigned char* output, int imgWidth, int imgHeight, int channels)	Fungsi rekursif untuk mengisi buffer gambar dari struktur QuadTree. Jika node adalah daun ( <i>leaf</i> ), maka warnanya digunakan untuk mengisi piksel sesuai area.
void reconstructAndSaveImage(const QuadTree& qt, const std::string& filename)	Fungsi untuk menyimpan gambar hasil rekonstruksi dari QuadTree ke file output.
bool isValidImage(unsigned char* image, int width, int height, int channels)	Fungsi untuk validasi apakah sebuah gambar memiliki format yang dapat digunakan.
void saveFrame(unsigned char* image, int width, int height, int channels, const std::string& folder)	Fungsi untuk menyimpan satu frame gambar dalam proses animasi. Digunakan untuk menciptakan urutan frame dalam folder sementara sebelum disatukan menjadi GIF.
void collectNodesPerDepth(QuadTreeNode* node, std::vector<std::vector<QuadTreeNode*>>& levels, int depth)	Fungsi untuk mengelompokkan node-node QuadTree berdasarkan kedalamannya (level) dalam vektor 2 dimensi, digunakan untuk animasi berbasis level rekonstruksi.

```
void reconstructByLevel(const QuadTree& qt, const std::string& gifPath)
```

Fungsi untuk membentuk animasi proses kompresi dengan pendekatan *level-order traversal*. Setiap level disimpan sebagai frame dan digabung menggunakan GIF.

## 2.5 File main.cpp

File ini berisi fungsi utama dari program.

Nama Fungsi	Deskripsi
<pre>QuadTree findBestThreshold(     const unsigned char* image,     int width, int height, int channels,     int errorMethod,     double targetRatio,     const std::string&amp; ext,     const std::string&amp; finalOutputPath,     int&amp; bestDepth,     int&amp; bestTotalNodes,     const std::string&amp; originalInputPath,     size_t originalSize )</pre>	<p>Fungsi yang bertugas menemukan nilai threshold terbaik secara iteratif (binary search) agar ukuran file hasil kompresi mendekati target yang ditentukan pengguna. Disimpan sementara di folder temp_quadtree.</p>
<pre>int main()</pre>	<p>Fungsi utama yang mengatur alur keseluruhan program. Mulai dari menerima input pengguna, memuat gambar, menjalankan algoritma kompresi, menyimpan hasil, dan menampilkan statistik.</p>

## BAB III

### SOURCE CODE

#### 2.1 IO.cpp

```
#include "header/IO.hpp"
#include <iostream>
#include <limits>
#include <algorithm>
#include "external-libs/stb_image.h"
#include "external-libs/stb_image_write.h"

using namespace std;

void printErrorMethodOptions() {
    cout << "Pilih metode error:\n";
    cout << "1 = Variance (>0)\n";
    cout << "2 = MAD (Mean Absolute Deviation) (0-255)\n";
    cout << "3 = Max Pixel Difference (0-255)\n";
    cout << "4 = Entropy (0-8)\n";
    cout << "5 = SSIM (0-1)\n";
}

InputParams getUserInput() {
    InputParams params;

    cout << "Masukkan path gambar input (absolute path): ";
    getline(cin, params.inputImagePath);

    size_t dotPos = params.inputImagePath.find_last_of('.');
    std::string inputExt = (dotPos != std::string::npos) ?
        params.inputImagePath.substr(dotPos + 1) : "";
    std::transform(inputExt.begin(), inputExt.end(), inputExt.begin(), ::tolower);

    printErrorMethodOptions();
    while (true) {
        cout << "Pilihanmu (1-5): ";
        cin >> params.errorMethod;
        if (params.errorMethod >= 1 && params.errorMethod <= 5) break;
        cout << "Metode tidak valid! Silakan pilih antara 1 sampai 5.\n";
    }

    while (true) {
        cout << "Masukkan threshold: ";
        cin >> params.threshold;
        if (params.threshold >= 0.0) break;
        cout << "Threshold tidak boleh negatif.\n";
    }

    while (true) {
        cout << "Masukkan ukuran blok minimum: ";
        cin >> params.minBlockSize;
        if (params.minBlockSize >= 0) break;
        cout << "Ukuran blok minimum tidak boleh negatif.\n";
    }
}
```

```

cout << "Masukkan target persentase kompresi (0 = nonaktifkan mode ini): ";
cin >> params.targetCompression;
cin.ignore(numeric_limits<streamsize>::max(), '\n');

while (true) {
    cout << "Masukkan path untuk gambar output (absolute path): ";
    getline(cin, params.outputImagePath);

    size_t dotOut = params.outputImagePath.find_last_of('.');
    std::string outputExt = (dotOut != std::string::npos) ?
        params.outputImagePath.substr(dotOut + 1) : "";
    std::transform(outputExt.begin(), outputExt.end(), outputExt.begin(), ::tolower);

    if (outputExt != inputExt) {
        cout << "Format output harus sama dengan input (" << inputExt << "). Silakan coba lagi.\n";
    } else {
        break;
    }
}
cout << "Masukkan path untuk menyimpan GIF (opsional, tekan enter jika tidak ingin): ";
getline(cin, params.outputGifPath);

return params;
}

void printOutputStats(const OutputStats& stats) {
    cout << "\n===== Statistik Kompresi =====\n";
    cout << "Waktu eksekusi      : " << stats.executionTime << " ms\n";
    cout << "Ukuran gambar sebelum : " << stats.originalSize << " bytes\n";
    cout << "Ukuran gambar setelah : " << stats.compressedSize << " bytes\n";
    cout << "Persentase kompresi   : " << (1-stats.compressionRatio)*100 << "%\n";
    cout << "Kedalaman pohon       : " << stats.maxDepth << "\n";
    cout << "Jumlah simpul pohon    : " << stats.totalNodes << "\n";
}

void printImageSavedMessage(const string& path) {
    cout << "Gambar hasil kompresi disimpan di: " << path << "\n";
}

void printGifSavedMessage(const string& path) {
    if (!path.empty()) {
        cout << "GIF proses kompresi disimpan di: " << path << "\n";
    }
}

unsigned char* load_image(const char* filename, int& width, int& height, int& channels) {
    unsigned char* data = stbi_load(filename, &width, &height, &channels, 0);
    if (!data) {
        cerr << "Gagal memuat gambar dari: " << filename << endl;
        exit(1);
    }
    return data;
}

```

## 2.2 ErrorCalculator.cpp

```
#include "header/ErrorCalculator.hpp"
#include <cmath>
#include <map>
#include <numeric>

namespace ErrorCalculator {
    double calculateMAD(const Block& block, int width, int height) {
        int channels = 3;
        int pixelCount = width * height;
        double mean[3] = {0.0, 0.0, 0.0};
        for (int i = 0; i < block.size(); i += channels) {
            for (int c = 0; c < channels; ++c) {
                mean[c] += block[i + c];
            }
        }
        for (int c = 0; c < channels; ++c) {
            mean[c] /= pixelCount;
        }
        double mad[3] = {0.0, 0.0, 0.0};
        for (int i = 0; i < block.size(); i += channels) {
            for (int c = 0; c < channels; ++c) {
                mad[c] += std::abs(static_cast<double>(block[i + c]) - mean[c]);
            }
        }
        for (int c = 0; c < channels; ++c) {
            mad[c] /= pixelCount;
        }

        return (mad[0] + mad[1] + mad[2]) / 3.0;
    }

    double calculateMaxDiff(const Block& block1, const Block& block2, int width, int height) {
        double maxDiff = 0.0;
        int size = width * height * 3;
        for (int i = 0; i < size; ++i) {
            double diff = std::abs(static_cast<double>(block1[i]) - static_cast<double>(block2[i]));
            if (diff > maxDiff) maxDiff = diff;
        }
        return maxDiff;
    }

    double calculateVariance(const Block& block, int width, int height) {
        const std::vector<uint8_t>& data = block;
        int pixelCount = width * height;
        double mean[3] = {0.0f, 0.0f, 0.0f};
        for (int i = 0; i < data.size(); i += 3) {
            mean[0] += data[i]; // R
            mean[1] += data[i + 1]; // G
            mean[2] += data[i + 2]; // B
        }
        mean[0] /= pixelCount;
        mean[1] /= pixelCount;
        mean[2] /= pixelCount;
        double variance[3] = {0.0f, 0.0f, 0.0f};
        for (int i = 0; i < data.size(); i += 3) {
            variance[0] += (data[i] - mean[0]) * (data[i] - mean[0]);
            variance[1] += (data[i + 1] - mean[1]) * (data[i + 1] - mean[1]);
            variance[2] += (data[i + 2] - mean[2]) * (data[i + 2] - mean[2]);
        }
        variance[0] /= pixelCount;
        variance[1] /= pixelCount;
        variance[2] /= pixelCount;

        return (variance[0] + variance[1] + variance[2]) / 3.0f;
    }
}
```

```

double calculateEntropy(const Block& block, int width, int height) {
    int channels = 3;
    int totalPixels = width * height;
    double entropy = 0.0;
    for (int c = 0; c < channels; ++c) {
        std::map<uint8_t, int> freq;
        for (int i = c; i < block.size(); i += channels) {
            freq[block[i]]++;
        }
        double channelEntropy = 0.0;
        for (const auto& [val, count] : freq) {
            double p = static_cast<double>(count) / totalPixels;
            channelEntropy -= p * std::log2(p);
        }
        entropy += channelEntropy;
    }
    return entropy / channels;
}

double calculateSSIM(const Block& block1, const Block& block2, int width, int height) {
    const double C1 = 6.5025, C2 = 58.5225;
    int channels = 3;
    int totalPixels = width * height;
    double ssim_total = 0.0;

    for (int c = 0; c < channels; ++c) {
        double mu1 = 0.0, mu2 = 0.0;
        for (int i = c; i < block1.size(); i += channels) {
            mu1 += block1[i];
            mu2 += block2[i];
        }
        mu1 /= totalPixels;
        mu2 /= totalPixels;

        double sigma1_sq = 0.0, sigma2_sq = 0.0, sigma12 = 0.0;
        for (int i = c; i < block1.size(); i += channels) {
            double a = block1[i] - mu1;
            double b = block2[i] - mu2;
            sigma1_sq += a * a;
            sigma2_sq += b * b;
            sigma12 += a * b;
        }

        sigma1_sq /= totalPixels;
        sigma2_sq /= totalPixels;
        sigma12 /= totalPixels;

        double ssim = ((2 * mu1 * mu2 + C1) * (2 * sigma12 + C2)) /
                     ((mu1 * mu1 + mu2 * mu2 + C1) * (sigma1_sq + sigma2_sq + C2));

        ssim_total += ssim;
    }
    return ssim_total / channels;
}

double calculateError(int methodId, const Block& block1, const Block& block2, int width, int height) {
    switch (methodId) {
        case MAD_METHOD:
            return calculateMAD(block1, width, height);
        case MAX_DIFF_METHOD:
            return calculateMaxDiff(block1, block2, width, height);
        case VARIANCE_METHOD:
            return calculateVariance(block1, width, height);
        case ENTROPY_METHOD:
            return calculateEntropy(block1, width, height);
        case SSIM_METHOD:
            return 1.0 - calculateSSIM(block1, block2, width, height); // SSIM tinggi = bagus, jadi kebalik
        default:
            return 0.0;
    }
}

```

## 2.3 QuadTree.cpp

```
#include "header/QuadTree.hpp"
#include <cmath>
#include <cstring>
#include <iostream>
#include <iomanip>
#include <queue>
#include <map>
#include <functional>

QuadTree::QuadTree(const unsigned char* img, int width, int height, int channels,
                   int minBlockSize, double threshold, int errorMethod)
    : image(img), imgWidth(width), imgHeight(height), channels(channels),
      minBlockSize(minBlockSize), threshold(threshold), errorMethod(errorMethod),
      maxDepth(0), totalNodes(0) {}

void QuadTree::buildTree() {
    root = build(0, 0, imgWidth, imgHeight, 0);
}

std::vector<uint8_t> QuadTree::getBlock(int x, int y, int width, int height) {
    std::vector<uint8_t> block(width * height * channels);
    for (int j = 0; j < height; ++j) {
        for (int i = 0; i < width; ++i) {
            for (int c = 0; c < channels; ++c) {
                block[(j * width + i) * channels + c] = image[((y + j) * imgWidth + (x + i)) * channels + c];
            }
        }
    }
    return block;
}

std::vector<uint8_t> QuadTree::calculateAverageColor(const std::vector<uint8_t>& block) {
    std::vector<uint64_t> sum(channels, 0);
    int totalPixels = block.size() / channels;
    for (size_t i = 0; i < block.size(); ++i) {
        sum[i % channels] += block[i];
    }
    std::vector<uint8_t> avg(channels);
    for (int c = 0; c < channels; ++c) {
        avg[c] = static_cast<uint8_t>(sum[c] / totalPixels);
    }
    return avg;
}

double QuadTree::calculateError(const std::vector<uint8_t>& block, const std::vector<uint8_t>& avgColor, int width, int height) {
    if (errorMethod == MAD_METHOD || errorMethod == VARIANCE_METHOD || errorMethod == ENTROPY_METHOD) {
        return ErrorCalculator::calculateError(errorMethod, block, block, width, height);
    }

    std::vector<uint8_t> avgBlock(block.size());
    for (size_t i = 0; i < block.size(); i += channels) {
        for (int c = 0; c < channels; ++c) {
            avgBlock[i + c] = avgColor[c];
        }
    }
    return ErrorCalculator::calculateError(errorMethod, block, avgBlock, width, height);
}

std::unique_ptr<QuadTreeNode> QuadTree::build(int x, int y, int width, int height, int depth) {
    auto node = std::make_unique<QuadTreeNode>(x, y, width, height);
    ++totalNodes;
    maxDepth = std::max(maxDepth, depth);

    auto block = getBlock(x, y, width, height);
    auto avgColor = calculateAverageColor(block);
    double error = calculateError(block, avgColor, width, height);
    node->color = avgColor;

    if (width <= minBlockSize || height <= minBlockSize || error <= threshold) {
        node->isLeaf = true;
        return node;
    }
}
```

```

node->isLeaf = false;
int halfW = width / 2;
int halfH = height / 2;

node->children[0] = build(x, y, halfW, halfH, depth + 1); // Top-left
node->children[1] = build(x + halfW, y, width - halfW, halfH, depth + 1); // Top-right
node->children[2] = build(x, y + halfH, halfW, height - halfH, depth + 1); // Bottom-left
node->children[3] = build(x + halfW, y + halfH, width - halfW, height - halfH, depth + 1); // Bottom-right

return node;
}

void QuadTree::collectNodesPerDepth(std::vector<std::vector<QuadTreeNode*>>& levels) {
    if (!root) return;

    std::queue<std::pair<QuadTreeNode*, int>> q;
    q.push({ root.get(), 0 });

    while (!q.empty()) {
        auto [node, depth] = q.front(); q.pop();

        if (levels.size() <= depth) {
            levels.emplace_back();
        }
        levels[depth].push_back(node);

        if (!node->isLeaf) {
            for (auto& child : node->children) {
                if (child) {
                    q.push({ child.get(), depth + 1 });
                }
            }
        }
    }
}

void QuadTree::collectNodesByDepth(std::map<int, std::vector<QuadTreeNode*>>& nodesByDepth) {
    std::function<void(QuadTreeNode*, int)> dfs = [&](QuadTreeNode* node, int depth) {
        if (!node) return;
        nodesByDepth[depth].push_back(node);
        if (node->isLeaf) return;
        for (auto& child : node->children) {
            if (child) dfs(child.get(), depth + 1);
        }
    };
    dfs(root.get(), 0);
}

int QuadTree::getMaxDepth() const {
    return maxDepth;
}

int QuadTree::getTotalNodes() const {
    return totalNodes;
}

const std::unique_ptr<QuadTreeNode>& QuadTree::getRoot() const {
    return root;
}

int QuadTree::getWidth() const { return imgWidth; }
int QuadTree::getHeight() const { return imgHeight; }
int QuadTree::getChannels() const { return channels; }

```

## 2.4 Reconstruct.cpp

```
#include "external-libs/stb_image_write.h"
#include "header/QuadTree.hpp"
#include <iostream>
#include <algorithm>
#include <filesystem>
#include <iomanip>
#include <sstream>
#include <cstdlib>
#include <vector>
#include <queue>

namespace fs = std::filesystem;

void reconstructNode(QuadTreeNode* node, unsigned char* output, int imgWidth, int imgHeight, int channels) {
    if (node->isLeaf) {
        for (int j = 0; j < node->height; ; ++j) {
            for (int i = 0; i < node->width; ; ++i) {
                int x = node->x + i;
                int y = node->y + j;
                if (x < imgWidth && y < imgHeight) {
                    int idx = (y * imgWidth + x) * channels;
                    for (int c = 0; c < channels; ++c) {
                        output[idx + c] = node->color[c];
                    }
                }
            }
        }
    } else {
        for (const auto& child : node->children) {
            if (child) {
                reconstructNode(child.get(), output, imgWidth, imgHeight, channels);
            }
        }
    }
}

void reconstructAndSaveImage(const QuadTree& qt, const std::string& filename) {
    int width = qt.getWidth();
    int height = qt.getHeight();
    int channels = qt.getChannels();

    unsigned char* outputImage = new unsigned char[width * height * channels];
    reconstructNode(qt.getRoot().get(), outputImage, width, height, channels);

    std::string ext = filename.substr(filename.find_last_of('.') + 1);
    std::transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
    bool success = false;

    if (ext == "png") {
        success = stbi_write_png(filename.c_str(), width, height, channels, outputImage, width * channels);
    } else if (ext == "jpg" || ext == "jpeg") {
        if (channels == 4) {
            std::cerr << "Warning: JPEG tidak mendukung alpha channel." << std::endl;
        }
        success = stbi_write_jpg(filename.c_str(), width, height, channels, outputImage, 100);
    } else {
        std::cerr << "Format tidak didukung (hanya PNG dan JPG)." << std::endl;
    }

    delete[] outputImage;
}

int frameCounter = 0;

bool isValidImage(unsigned char* image, int width, int height, int channels) {
    return image && width > 0 && height > 0 && channels > 0;
}
```

```
void saveFrame(unsigned char* image, int width, int height, int channels, const std::string& folder) {
    if (!isValidImage(image, width, height, channels)) return;

    std::ostringstream ss;
    ss << folder << "/frame_" << std::setw(4) << std::setfill('0') << frameCounter++ << ".png";
    std::string filename = ss.str();

    if (!stbi_write_png(filename.c_str(), width, height, channels, image, width * channels)) {
        std::cerr << "Gagal menyimpan frame: " << filename << std::endl;
        std::remove(filename.c_str());
    }
}

void collectNodesPerDepth(QuadTreeNode* node, std::vector<std::vector<QuadTreeNode*>>& levels, int depth) {
    if (!node) return;
    if (depth >= levels.size()) levels.resize(depth + 1);
    levels[depth].push_back(node);
    if (!node->isLeaf) {
        for (const auto& child : node->children) {
            collectNodesPerDepth(child.get(), levels, depth + 1);
        }
    }
}
```

```

void reconstructByLevel(const QuadTree& qt, const std::string& gifPath) {
    int width = qt.getWidth();
    int height = qt.getHeight();
    int channels = qt.getChannels();

    if (width <= 0 || height <= 0 || channels <= 0) {
        std::cerr << "Dimensi atau channel gambar tidak valid." << std::endl;
        return;
    }

    unsigned char* image = new unsigned char[width * height * channels];
    std::fill(image, image + width * height * channels, 255);

    fs::path gifFilePath(gifPath);
    fs::path gifDir = gifFilePath.parent_path();
    std::string frameFolder = (gifDir / "frames").string();

    try {
        fs::create_directories(frameFolder);
    } catch (const fs::filesystem_error& e) {
        std::cerr << "Gagal membuat folder frame: " << e.what() << std::endl;
        delete[] image;
        return;
    }

    frameCounter = 0;

    std::queue<QuadTreeNode*> q;
    q.push(qt.getRoot().get());

    while (!q.empty()) {
        int levelSize = q.size();

        for (int i = 0; i < levelSize; ++i) {
            QuadTreeNode* node = q.front(); q.pop();

            std::vector<uint8_t>& color = node->color;

            for (int j = 0; j < node->height; height; ++j) {
                for (int i = 0; i < node->width; width; ++i) {
                    int x = node->x + i;
                    int y = node->y + j;
                    if (x < width && y < height) {
                        int idx = (y * width + x) * channels;
                        for (int c = 0; c < channels; ++c) {
                            image[idx + c] = color[c];
                        }
                    }
                }
            }

            if (!node->isLeaf) {
                for (const auto& child : node->children) {
                    if (child) q.push(child.get());
                }
            }
        }

        saveFrame(image, width, height, channels, frameFolder);
    }

    std::string command = "magick -delay 50 -loop 0 \"" + frameFolder + "/frame_*.png\" \\" + gifPath + "\"";
    std::cout << "Membuat GIF: " << command << std::endl;
    int result = std::system(command.c_str());

    if (result != 0) {
        std::cerr << "Gagal membuat GIF. Pastikan ImageMagick tersedia." << std::endl;
    } else {
        std::cout << "GIF disimpan: " << gifPath << std::endl;
        try {
            fs::remove_all(frameFolder);
        } catch (const fs::filesystem_error& e) {
            std::cerr << "Gagal menghapus folder frame: " << e.what() << std::endl;
        }
    }

    delete[] image;
}

```

## 2.5 main.cpp

```
#include <iostream>
#include <chrono>
#include <filesystem>
#include <algorithm>
#include "header/IO.hpp"
#include "header/QuadTree.hpp"
#include "header/Reconstruct.hpp"
#include "external-libs/stb_image.h"
#include "external-libs/stb_image_write.h"

namespace fs = std::filesystem;
using namespace std;
using namespace std::chrono;

QuadTree findBestThreshold(
    const unsigned char* image,
    int width, int height, int channels,
    int errorMethod,
    double targetRatio,
    const std::string& ext,
    const std::string& finalOutputPath,
    int& bestDepth,
    int& bestTotalNodes,
    const std::string& originalInputPath,
    size_t originalSize
) {
    const std::string tempFolder = "temp_quadtree";
    fs::create_directories(tempFolder);

    double low = 0.0, high = 100.0;
    switch (errorMethod) {
        case 1: // Variance
            high = 20000.0;
            break;
        case 2: // MAD
            high = 255.0;
            break;
        case 3: // Max Pixel Difference
            high = 255.0;
            break;
        case 4: // Entropy
            low = 0.5;
            high = 8.0;
            break;
        case 5: // SSIM
            high = 1.0;
            break;
        default:
            std::cerr << "Metode error tidak valid. Gunakan default range.\n";
            break;
    }
    double bestDiff = 1.0;
    QuadTree bestTree(nullptr, 0, 0, 0, 0, 0, 0);
    std::string bestPath = tempFolder + "/best_output." + ext;
    bool hasBest = false;
```

```
for (int i = 0; i < 10; ++i) {
    double mid = (low + high) / 2.0;
    // cout << "Low : " << low << endl;
    // cout << "High : " << high << endl;
    QuadTree qt(image, width, height, channels, 1, mid, errorMethod);
    qt.buildTree();

    std::string trialPath = tempFolder + "/trial_" + std::to_string(i) + "." + ext;
    reconstructAndSaveImage(qt, trialPath);

    size_t size = fs::file_size(trialPath);
    // cout << "Temp size : " << size << endl;
    double ratio = static_cast<double>(size) / originalSize;
    // cout << "Ratio : " << ratio << endl;
    double diff = abs(ratio - targetRatio);
    // cout << "Diff : " << diff << endl;

    if (diff < bestDiff) {
        bestDiff = diff;
        bestTree = std::move(qt);
        bestDepth = bestTree.getMaxDepth();
        bestTotalNodes = bestTree.getTotalNodes();
        try {
            if (fs::exists(bestPath)) {
                fs::remove(bestPath);
            }
            fs::copy_file(trialPath, bestPath);
            hasBest = true;
        } catch (const fs::filesystem_error& e) {
            std::cerr << "Gagal menyimpan best output: " << e.what() << std::endl;
        }
    }

    if (ratio > targetRatio) low = mid;
    else high = mid;
}

if (hasBest && fs::exists(bestPath)) {
    int w, h, ch;
    unsigned char* data = stbi_load(bestPath.c_str(), &w, &h, &ch, 0);
    if (!data) {
        std::cerr << "Gagal memuat gambar dari bestPath untuk disalin ke output akhir." << std::endl;
    } else {
        bool success = false;
```

```

        if (ext == "png") {
            success = stbi_write_png(finalOutputPath.c_str(), w, h, ch, data, w * ch);
        } else if (ext == "jpg" || ext == "jpeg") {
            if (ch == 4) {
                std::cerr << "Warning: JPEG tidak mendukung alpha channel.\n";
            }
            success = stbi_write_jpg(finalOutputPath.c_str(), w, h, ch, data, 100);
        } else {
            std::cerr << "Format tidak didukung: hanya PNG, JPG, dan JPEG.\n";
        }

        if (!success) {
            std::cerr << "Gagal menulis hasil akhir dengan STB Image.\n";
        }

        stbi_image_free(data);
    }

    try {
        fs::remove_all(tempFolder);
    } catch (...) {}

    return bestTree;
}

int main() {
    InputParams params = getUserInput();

    int width, height, channels;
    unsigned char* image = load_image(params.inputImagePath.c_str(), width, height, channels);
    if (!image) {
        cerr << "Gagal memuat gambar dari: " << params.inputImagePath << endl;
        return 1;
    }

    std::string ext = fs::path(params.inputImagePath).extension().string();
    std::transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
    if (!ext.empty() && ext[0] == '.') ext = ext.substr(1);

    size_t originalSize = fs::file_size(params.inputImagePath);
    QuadTree qt(nullptr, 0, 0, 0, 0, 0.0, 0);
    int maxDepth = 0, totalNodes = 0;

    auto start = high_resolution_clock::now();

    if (params.targetCompression > 0.0) {
        qt = findBestThreshold(
            image, width, height, channels,
            params.errorMethod,
            1 - params.targetCompression,
            ext,
            params.outputImagePath,
            maxDepth,
            totalNodes,
            params.inputImagePath,
            originalSize
        );
        maxDepth = qt.getMaxDepth();
        totalNodes = qt.getTotalNodes();
    } else {
        qt = QuadTree(image, width, height, channels, params.minBlockSize, params.threshold, params.errorMethod);
        qt.buildTree();
        reconstructAndSaveImage(qt, params.outputImagePath);
        maxDepth = qt.getMaxDepth();
        totalNodes = qt.getTotalNodes();
    }

    auto end = high_resolution_clock::now();
    auto executionTime = duration_cast<milliseconds>(end - start).count();

    printImageSavedMessage(params.outputImagePath);

    if (!params.outputGifPath.empty()) {
        reconstructByLevel(qt, params.outputGifPath);
    }
}

```

```
OutputStats stats {
    static_cast<double>(executionTime),
    originalSize,
    compressedSize,
    compressedSize / static_cast<double>(originalSize),
    maxDepth,
    totalNodes
};

printOutputStats(stats);
stbi_image_free(image);

return 0;
}
```

## BAB IV

### TESTING

#### 3.1 Metode Variance

Input

Kasus 1 (Threshold 200, Min Block 8)
<pre>===== Statistik Kompresi ===== Waktu eksekusi      : 1172 ms Ukuran gambar sebelum : 1851132 bytes Ukuran gambar setelah : 2361523 bytes Persentase kompresi    : -27.5718% Kedalaman pohon       : 9 Jumlah simpul pohon   : 60585</pre>



GIF :

[https://drive.google.com/file/d/1-GZureoOwEgt9sBBShZt6buQvHXMAWIH/view?usp=share\\_link](https://drive.google.com/file/d/1-GZureoOwEgt9sBBShZt6buQvHXMAWIH/view?usp=share_link)

**Kasus 2 (Threshold 50, Min Block 8)**

```
===== Statistik Kompresi =====
Waktu eksekusi      : 1264 ms
Ukuran gambar sebelum : 1851132 bytes
Ukuran gambar setelah : 2688084 bytes
Persentase kompresi    : -45.213%
Kedalaman pohon       : 9
Jumlah simpul pohon   : 89585
```



GIF :

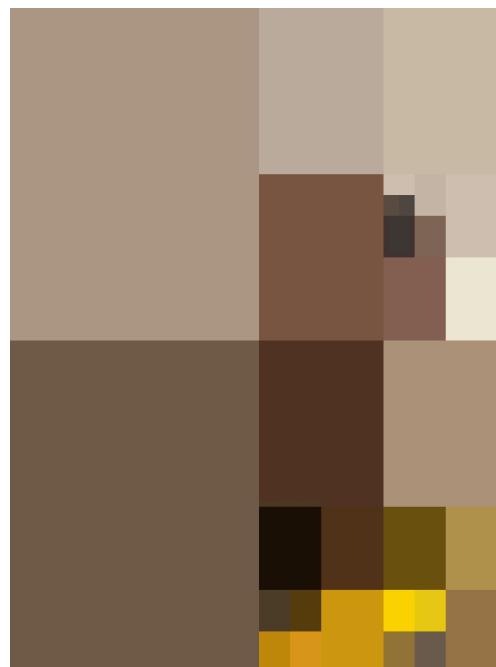
[https://drive.google.com/file/d/1-GZureoOwEgt9sBBShZt6buQvHXMAWIH/view?usp=share\\_link](https://drive.google.com/file/d/1-GZureoOwEgt9sBBShZt6buQvHXMAWIH/view?usp=share_link)

### 3.2 Metode Mean Absolute Deviation

Input
A photograph of a young man with dark hair, wearing a brown zip-up jacket over a black shirt. He is looking down at a yellow rectangular object he is holding in his hands. In the background, another person is visible, also looking down at something. The setting appears to be indoors.

### Kasus 1 (Threshold 50, Min Block 4)

```
===== Statistik Kompresi =====
Waktu eksekusi      : 534 ms
Ukuran gambar sebelum : 1586288 bytes
Ukuran gambar setelah : 385110 bytes
Persentase kompresi    : 75.7226%
Kedalaman pohon       : 5
Jumlah simpul pohon   : 41
```



GIF :

[https://drive.google.com/file/d/1jDtkOrr-xeYkmaLA5K1CNKKhRh46PLvC/view?usp=share\\_link](https://drive.google.com/file/d/1jDtkOrr-xeYkmaLA5K1CNKKhRh46PLvC/view?usp=share_link)

### Kasus 2 (Threshold 10, Min Block 4)

```
===== Statistik Kompresi =====
Waktu eksekusi      : 1096 ms
Ukuran gambar sebelum : 1586288 bytes
Ukuran gambar setelah : 2094391 bytes
Persentase kompresi    : -32.0309%
Kedalaman pohon       : 9
Jumlah simpul pohon   : 41389
```



GIF :

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

### 3.3 Metode Max Pixel Difference

Input
A photograph of a person with glasses, wearing a black t-shirt with the number '196' on it, standing with their arms crossed. They are smiling and looking towards the camera. The background shows a wall with a sign that partially reads 'FOR EVERYONE'.
Kasus 1 (Threshold 100, Min Block 8)

```
===== Statistik Kompresi =====
Waktu eksekusi : 964 ms
Ukuran gambar sebelum : 1615966 bytes
Ukuran gambar setelah : 937012 bytes
Persentase kompresi : 42.0154%
Kedalaman pohon : 9
Jumlah simpul pohon : 7189
```



GIF : (3\_1.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

#### Kasus 2 (Threshold 50, Min Block 8)

```
===== Statistik Kompresi =====
Waktu eksekusi : 1129 ms
Ukuran gambar sebelum : 1615966 bytes
Ukuran gambar setelah : 1339079 bytes
Persentase kompresi : 17.1345%
Kedalaman pohon : 9
Jumlah simpul pohon : 17041
```



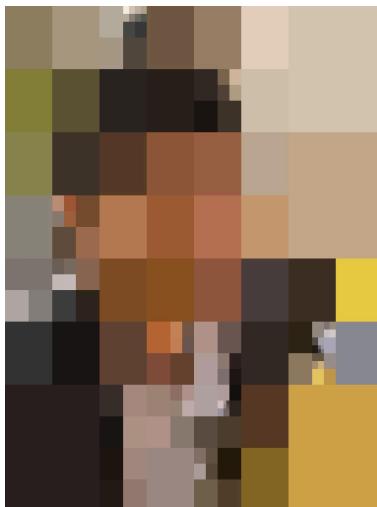
GIF : (3\_2.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

### 3.4 Metode Entropy

Input
<b>Kasus 1 (Threshold 7, Min Block 8)</b>

```
===== Statistik Kompresi =====
Waktu eksekusi      : 3905 ms
Ukuran gambar sebelum : 2040245 bytes
Ukuran gambar setelah : 500359 bytes
Persentase kompresi    : 75.4755%
Kedalaman pohon       : 7
Jumlah simpul pohon   : 197
```



GIF : (4\_1.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

#### Kasus 2 (Threshold 1, Min Block 8)

```
===== Statistik Kompresi =====
Waktu eksekusi      : 10166 ms
Ukuran gambar sebelum : 2040245 bytes
Ukuran gambar setelah : 3670931 bytes
Persentase kompresi    : -79.926%
Kedalaman pohon       : 9
Jumlah simpul pohon   : 349525
```



GIF : (4\_2.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

### 3.5 Metode Structure Similarity Index

Input
<b>Kasus 1 (Threshold 0.8, Min Block 8)</b>

```
===== Statistik Kompresi =====
Waktu eksekusi : 994 ms
Ukuran gambar sebelum : 1168416 bytes
Ukuran gambar setelah : 1221352 bytes
Persentase kompresi : -4.53058%
Kedalaman pohon : 9
Jumlah simpul pohon : 21389
```



GIF : (5\_1.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

#### Kasus 2 (Threshold 0.2, Min Block 8)

```
===== Statistik Kompresi =====
Waktu eksekusi : 1426 ms
Ukuran gambar sebelum : 1168416 bytes
Ukuran gambar setelah : 2293878 bytes
Persentase kompresi : -96.3237%
Kedalaman pohon : 9
Jumlah simpul pohon : 164301
```



GIF : (5\_2.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

### 3.6 Black and White

Input
A black and white close-up photograph of a lion's face, showing its eyes, nose, and mouth.
Kasus 1 (Variance, Threshold 50, Min Block 4)
<pre>===== Statistik Kompresi ===== Waktu eksekusi      : 47 ms Ukuran gambar sebelum  : 117223 bytes Ukuran gambar setelah   : 160005 bytes Persentase kompresi     : -36.4963% Kedalaman pohon       : 7 Jumlah simpul pohon    : 20545</pre>



GIF : (6\_1.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzQ>

### 3.7 Target Kompresi

Input
<b>Kasus 1 (Target 0.8, Metode Variance)</b>

```
===== Statistik Kompresi =====
Waktu eksekusi : 2441 ms
Ukuran gambar sebelum : 1157694 bytes
Ukuran gambar setelah : 251090 bytes
Persentase kompresi : 78.3112%
Kedalaman pohon : 5
Jumlah simpul pohon : 29
```

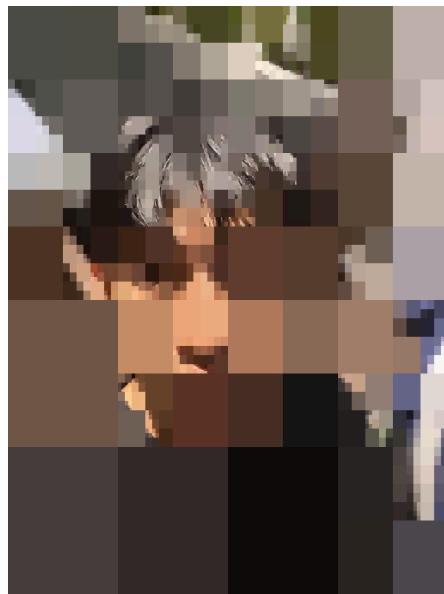


GIF : (7\_1.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzO>

#### Kasus 2 (Target 0.5, Metode SSIM)

```
===== Statistik Kompresi =====
Waktu eksekusi : 8929 ms
Ukuran gambar sebelum : 1157694 bytes
Ukuran gambar setelah : 597968 bytes
Persentase kompresi : 48.3484%
Kedalaman pohon : 12
Jumlah simpul pohon : 3521
```



GIF : (7\_2.gif)

<https://drive.google.com/drive/folders/1gmnG-4Xrgww7E0Kf4H8CxOkY9Qp3zFzQ>

## BAB V

### IMPLEMENTASI BONUS

#### **5.1 Bonus Target Persentase Kompresi**

Secara deskripsi, bonus ini memungkinkan pengguna menentukan target persentase kompresi berupa nilai floating point (contoh: 1.0 = 100%). Jika pengguna memberikan nilai 0, mode ini akan dinonaktifkan. Ketika mode ini diaktifkan, algoritma akan menyesuaikan nilai threshold secara otomatis untuk mencapai target persentase kompresi yang ditentukan. Penyesuaian threshold ini dilakukan secara dinamis, memberikan fleksibilitas dalam proses kompresi untuk memenuhi target efisiensi sambil mempertahankan kualitas gambar.

Dalam implementasi kode dari program yang telah dibuat, target presentasi kompresi utamanya tercantum dalam fungsi `findBestThreshold( const unsigned char* image, int width, int height, int channels, int errorMethod, double targetRatio, const std::string& ext, const std::string& finalOutputPath, int& bestDepth, int& bestTotalNodes, const std::string& originalInputPath, size_t originalSize)`. Tujuan dari fitur ini adalah agar pengguna dapat menentukan ukuran akhir relatif terhadap ukuran gambar asli, tanpa perlu melakukan trial-and-error manual dalam menentukan nilai threshold.

Cara kerjanya mulai dari fungsi akan menjalankan pencarian selama 10 iterasi. Pada setiap iterasi, nilai threshold yang tengah (`mid`) akan digunakan untuk membangun struktur *QuadTree* dan menghasilkan gambar kompresi sementara yang disimpan ke dalam file temporer. Ukuran file hasil kompresi ini dihitung, dan dibandingkan dengan ukuran gambar asli untuk memperoleh rasio kompresi. Selanjutnya, perbedaan antara rasio saat ini dan target akan dihitung, dan threshold yang menghasilkan perbedaan terkecil akan disimpan sebagai kandidat terbaik (`bestTree`). Apabila rasio saat ini lebih besar dari target, maka batas bawah pencarian akan digeser ke atas (threshold dinaikkan), sebaliknya jika terlalu kecil maka batas atas diturunkan, sesuai prinsip binary search. Setelah iterasi selesai, hasil terbaik yang paling mendekati target kompresi akan digunakan untuk menyimpan gambar akhir, dan *QuadTree* dari kompresi terbaik dikembalikan. Dengan demikian, fitur ini tidak hanya membantu pengguna mencapai ukuran file yang lebih sesuai kebutuhan, tetapi juga memanfaatkan pendekatan algoritmik yang efisien dan dapat digunakan secara universal untuk semua metode error yang tersedia.

## **5.2 Bonus Structure Similarity Index (SSIM)**

Secara deskripsi, bonus ini adalah metode pengukuran error dengan menghitung SSIM untuk setiap kanal warna R, G, dan B, lalu gabungkan hasilnya menjadi SSIM total menggunakan bobot masing-masing kanal.

Dalam implementasi kode yang telah dibuat, perhitungan error menggunakan metode SSIM tercantum dalam fungsi bernama calculateSSIM(`const Block& block1, const Block& block2, int width, int height`). Proses penghitungan SSIM dilakukan per kanal warna (Red, Green, Blue) dalam fungsi calculateSSIM. Mula-mula, dihitung rata-rata ( $\mu$ ) dari intensitas warna masing-masing blok untuk setiap kanal, lalu dilanjutkan dengan perhitungan variansi dan kovariansi antar blok tersebut. Hasil dari perhitungan tersebut dimasukkan ke dalam rumus SSIM yang melibatkan dua konstanta stabilisasi (C1 dan C2) untuk mencegah pembagian nol. Nilai SSIM masing-masing kanal kemudian dirata-ratakan untuk memperoleh SSIM akhir antar blok. Jika nilai SSIM lebih tinggi daripada nilai ambang (threshold) yang ditentukan pengguna, maka blok dianggap cukup seragam dan tidak perlu dibagi lebih lanjut. Sebaliknya, jika nilai SSIM lebih rendah dari ambang, maka blok akan dibagi menjadi empat bagian dan proses evaluasi diulang pada masing-masing bagian.

## **5.3 Bonus GIF**

Secara deskripsi, bonus ini memungkinkan pengguna memvisualisasikan proses pembentukan Quadtree dalam kompresi gambar dengan format GIF.

Dalam implementasi pada program ini, GIF utamanya dibentuk dari fungsi bernama `reconstructByLevel(const QuadTree& qt, const std::string& gifPath)`. Fungsi utama yang menangani pembuatan GIF adalah `reconstructByLevel`, yang menggunakan pendekatan level-order traversal (traversal per kedalaman pohon). Pada awalnya, sebuah canvas kosong berwarna putih disiapkan sebagai latar belakang. Kemudian, proses rekonstruksi dilakukan per level pohon Quadtree, artinya simpul-simpul (blok-blok gambar) yang berada pada kedalaman yang sama akan digambar bersamaan ke dalam frame yang sama. Hal ini membuat setiap frame dalam GIF merepresentasikan kemajuan visual pada tingkat kedalaman tertentu dari kompresi, mulai dari generalisasi kasar (misalnya seluruh gambar diwakili oleh satu warna), hingga mendekati bentuk asli gambar saat detail semakin banyak ditambahkan.

## **BAB VI**

### **ANALISIS PROGRAM**

Program kompresi gambar berbasis QuadTree ini terdiri dari beberapa tahapan utama dengan kompleksitas waktu  $O(n \log n)$ , di mana  $n$  menyatakan jumlah total piksel pada gambar ( $\text{width} \times \text{height}$ ). Pertama, program meminta input dari pengguna berupa path absolut gambar yang akan diproses, lalu memuat gambar tersebut ke dalam bentuk matriks piksel satu dimensi. Proses pemuatannya memiliki kompleksitas  $O(n)$  karena setiap piksel dalam gambar harus dibaca dan disimpan dalam memori.

Tahapan inti dari program adalah proses kompresi menggunakan struktur data QuadTree. Kompresi dilakukan dengan pendekatan rekursif divide-and-conquer, di mana gambar dibagi menjadi blok-blok yang lebih kecil secara berulang, sampai blok tersebut memenuhi kriteria homogenitas atau ukuran minimumnya tercapai. Pada setiap simpul dalam pohon, algoritma melakukan tiga operasi utama: mengambil blok piksel ( $O(n)$ ), menghitung warna rata-rata ( $O(n)$ ), dan menghitung nilai error berdasarkan metode yang dipilih ( $O(n)$ ). Jika nilai error masih melebihi threshold, blok akan dibagi menjadi empat kuadran dan proses rekursif dilanjutkan ke tiap bagian. Berdasarkan sifat ini, kompleksitas waktu algoritma mengikuti bentuk rekurrensi  $T(n) = 4T(n/4) + O(n)$ , yang jika dianalisis menggunakan Teorema Master akan menghasilkan  $T(n) = O(n \log n)$ .

Program menyediakan lima metode perhitungan error: Variance, Mean Absolute Deviation (MAD), Max Pixel Difference, Entropy, dan SSIM. Semua metode ini bekerja dalam kompleksitas  $O(n)$  per blok karena memerlukan iterasi menyeluruh terhadap piksel untuk menghitung statistik atau distribusi nilai warna. Dengan demikian, pemilihan metode error tidak mengubah kompleksitas keseluruhan kompresi dari  $O(n \log n)$ .

Jika pengguna mengaktifkan mode pencarian threshold otomatis berdasarkan target rasio kompresi, program akan menjalankan fungsi pencarian `findBestThreshold` sebanyak 10 iterasi. Dalam setiap iterasi, program membangun ulang pohon QuadTree dan mengevaluasi hasil kompresi—masing-masing membutuhkan  $O(n \log n)$ . Karena jumlah iterasi tetap, kompleksitas total dari pencarian threshold ini tetap dalam batas  $O(n \log n)$ .

Setelah proses kompresi selesai, program akan menyusun ulang gambar dari struktur QuadTree ke dalam format file. Proses ini mengisi kembali seluruh matriks piksel berdasarkan

simpul leaf pohon dan menyimpannya ke dalam file PNG atau JPG. Tahapan ini juga bekerja dalam kompleksitas  $O(n)$ .

Jika pengguna menginginkan output berupa GIF dari proses kompresi, program akan menyimpan gambar pada setiap level kedalaman pohon ke dalam frame-frame terpisah, lalu menggabungkannya menggunakan ImageMagick. Karena kedalaman pohon maksimum adalah logaritmik terhadap ukuran gambar, proses ini memiliki kompleksitas  $O(n \log n)$  dan bersifat opsional.

Jadi, secara keseluruhan program ini memiliki kompleksitas waktu  $O(n \log n)$  dengan kompleksitas ruang  $O(n)$ .

## **BAB VII**

### **KESIMPULAN**

Program ini merepresentasikan penerapan algoritma *divide and conquer* dalam permasalahan kompresi gambar melalui struktur data QuadTree. Proses dimulai dengan membagi gambar menjadi blok-blok kecil dan mengevaluasi homogenitasnya menggunakan metode perhitungan error seperti Variance, MAD, Max Pixel Difference, Entropy, dan SSIM. Jika blok tidak memenuhi kriteria homogenitas berdasarkan threshold yang diberikan, blok akan dibagi lagi menjadi empat bagian secara rekursif. Pendekatan ini membentuk struktur pohon di mana setiap simpul mewakili suatu wilayah gambar dan proses pembagian dilakukan hingga kondisi leaf tercapai. Secara umum, proses ini mengikuti pola rekurensi  $T(n) = 4T(n/4) + O(n)$ , yang berdasarkan Teorema Master menghasilkan kompleksitas waktu  $O(n \log n)$ . Seluruh tahap, mulai dari pembacaan input, pembentukan pohon quadtree, perhitungan nilai error, hingga rekonstruksi gambar hasil kompresi, merupakan rangkaian implementasi langkah-langkah divide and conquer. Dengan demikian, program ini menampilkan bagaimana algoritma berbasis pembagian rekursif dapat diimplementasikan untuk menyelesaikan permasalahan kompresi.

## LAMPIRAN

### Repository Program

Repository program dapat diakses melalui tautan GitHub berikut :

[https://github.com/fithrarzk/Tucil2\\_13523049\\_13523118](https://github.com/fithrarzk/Tucil2_13523049_13523118)

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	

## **DAFTAR PUSTAKA**

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf)

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