

**Laporan Tugas Kecil 2
IF2211 Strategi Algoritma**

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh:

Muhammad Timur Kanigara - 13523055

Aliya Husna Fayyaza - 13523062

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

DAFTAR ISI

BAB 1 ALGORITMA DIVIDE AND CONQUER	2
BAB 2 SOURCE CODE	4
BAB 3 TEST CASE	22
BAB 4 HASIL DAN ANALISIS	24
4.1 Hasil dan Analisis	24
4.2 Kompleksitas Algoritma	24
4.3 Kesimpulan	25
BAB 5 PENJELASAN ALGORITMA BONUS	26
5.1 Implementasi Structural Similarity Index Measure (SSIM)	26
5.2 Visualisasi Proses Pembentukan Quadtree dengan Format GIF	26
LAMPIRAN	28

BAB 1

ALGORITMA DIVIDE AND CONQUER

```
Function buildQuadtree()

function buildQuadtree(image, node, threshold, min_size, errorMethod, createGIF)
    if node is null then
        return

    x ← node.x
    y ← node.y
    width ← node.width
    height ← node.height
    node.avgColor ← avgColor(image, x, y, width, height)

    if errorMethod = 1 then
        var ← calVariance(image, x, y, width, height)
    else if errorMethod = 2 then
        var ← calMAD(image, x, y, width, height)
    else if errorMethod = 3 then
        var ← calMaxDiff(image, x, y, width, height)
    else if errorMethod = 4 then
        var ← calEntropy(image, x, y, width, height)
    else if errorMethod = 5 then
        avgBlock ← matrix(height, width) filled with node.avgColor
        var ← 1.0 - calSSIM(image, avgBlock, x, y, width, height)
    else
        print "Error: metode tidak dikenal"
        return

    isLeaf ← (var ≤ threshold or width ≤ min_size or height ≤ min_size)
    node.isLeaf ← isLeaf

    if not isLeaf then
        halfWidth ← width / 2
        halfHeight ← height / 2

        if halfWidth × halfHeight < min_size then
            node.isLeaf ← true
            return

        node.children[0] ← createNode(x, y, halfWidth, halfHeight)
        node.children[1] ← createNode(x + halfWidth, y, width - halfWidth, halfHeight)
        node.children[2] ← createNode(x, y + halfHeight, halfWidth, height - halfHeight)
        node.children[3] ← createNode(x + halfWidth, y + halfHeight, width - halfWidth, height
        - halfHeight)

        for i from 0 to 3 do
            buildQuadtree(image, node.children[i], threshold, min_size, errorMethod, createGIF)
```

Bagian dari program yang mengandung algoritma *divide and conquer* adalah fungsi *buildQuadTree* yang berperan sebagai algoritma penyusun pohon yang menerima input matriks vektor

RGB dari gambar yang ingin dikompresi serta beberapa parameter pendukung lain seperti koordinat, *threshold*, ukuran minimum pecahan gambar, serta *tree* yang digunakan sebagai penyimpan struktur pohonnya itu sendiri. Program dimulai dengan menghitung error (sesuai dengan metode pilihan pengguna) dari blok saat ini lalu menginisialisasi simpul baru (berikut koordinat, ukuran, dan *boolean* yang menandakan simpul tersebut merupakan daun atau bukan). Program lalu menginisialisasi 4 anak untuk tiap simpul dengan nilai -1 sebagai penanda belum ada anak. Selanjutnya, program juga menghitung rata-rata warna untuk kepentingan normalisasi warna. Selanjutnya, program menyimpan simpul ke dalam struktur pohon.

Inti dari algoritma *divide and conquer* terletak pada blok kode yang diawali dengan kode ‘*if not isLeaf*’ yang melakukan rekursi ke 4 pembagian gambar jika simpul bukan merupakan daun. Ukuran akan dibagi sama rata untuk tiap anak/*node* yang berada di level yang sama (dibagi 2 bukan 4 karena pembagian dilakukan terhadap *width* dan *length* untuk menjaga bentuk node sama seperti gambar asli, sehingga efeknya akan sama dengan membagi 4 luas gambar pada iterasi sebelumnya). Selanjutnya, dilakukan pemanggilan fungsi kembali (rekursi) untuk tiap anak dengan parameter yang disesuaikan untuk tiap posisi anaknya, namun tetap disimpan ke dalam variabel *image* agar didapatkan hasil akhir berupa pohon yang menyimpan seluruh simpul. Untuk penjelasan lebih lanjut, berikut adalah tahapannya.

1. Divide: Fungsi memeriksa nilai error dari suatu blok gambar menggunakan metode dan *threshold* yang dipilih pengguna. Fungsi juga mempertimbangkan ukuran minimum dari tiap simpul yang juga merupakan *input* dari pengguna. Jika nilai error melebihi *threshold* dan ukuran blok masih lebih besar dari ukuran minimum (termasuk jika blok dibagi 4), maka blok tersebut akan dibagi menjadi empat sub-blok berukuran seperempatnya.
2. Conquer: Untuk tiap sub-blok yang telah dibagi, fungsi dipanggil kembali secara rekursif dengan parameter yang sesuai. Proses ini akan terus berlangsung hingga seluruh blok memenuhi salah satu kondisi penghentian pembagian (nilai error di bawah *threshold* atau ukuran blok di bawah ukuran minimum)
3. Combine: Setelah seluruh blok diproses, simpul-simpul pohon disusun secara hierarkis (terbagi per level). Setiap simpul menyimpulkan rata-rata warna untuk blok tersebut serta hubungan antara simpul anak dan orang tua direpresentasikan melalui struktur pohon yang dibentuk selama proses rekursi (disimpan dalam larik berisi *pointer* ke simpul anak yang dimiliki oleh tiap simpul).

Untuk rekonstruksi gambar berdasarkan pohon yang terbentuk, akan dilakukan iterasi untuk membuat kembali matriks vektor RGB dari rata-rata warna RGB yang dimiliki oleh tiap simpul.

BAB 2

SOURCE CODE

```
errorcounter.h
```

```
#ifndef ERRORCOUNTER_H
#define ERRORCOUNTER_H

#include <vector>
#include <array>
#include <string>
#include "inout.h"
#include "quadtree.h"

double calVariance(const std::vector<std::vector<RGB>>& img, int x, int y, int width, int height);

double calMAD(const std::vector<std::vector<RGB>>& img, int x, int y, int width, int height);

double calMaxDiff(const std::vector<std::vector<RGB>>& img, int x, int y, int width, int height);

double calEntropy(const std::vector<std::vector<RGB>>& img, int x, int y, int width, int height);

double calSSIM(const std::vector<std::vector<RGB>>& imgRef, const
std::vector<std::vector<RGB>>& img, int x, int y, int width, int height);

#endif
```

```
errorcounter.cpp
```

```
#include <iostream>
#include <map>
#include <cmath>
#include <fstream>
#include <queue>
#include <FreeImage.h>
#include "headers/inout.h"
#include "headers/errorcounter.h"

using namespace std;
```

```

// Hitung variance dalam blok RGB
double calVariance(const vector<vector<RGB>>& img, int x, int y, int width, int height) {
    long totR = 0, totG = 0, totB = 0;
    int N = width * height;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            totR += img[i][j][0];
            totG += img[i][j][1];
            totB += img[i][j][2];
        }
    }

    double avgR = totR / (double)N;
    double avgG = totG / (double)N;
    double avgB = totB / (double)N;

    double var = 0.0;
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            var += pow(img[i][j][0] - avgR, 2);
            var += pow(img[i][j][1] - avgG, 2);
            var += pow(img[i][j][2] - avgB, 2);
        }
    }

    return var / N;
}

// Hitung MAD dalam blok RGB
double calMAD(const vector<vector<RGB>>& img, int x, int y, int width, int height) {
    long totR = 0, totG = 0, totB = 0;
    int N = width * height;

    for (int i = y; i < y + height && i < img.size(); i++) {
        for (int j = x; j < x + width && j < img[0].size(); j++) {
            totR += img[i][j][0];
            totG += img[i][j][1];
            totB += img[i][j][2];
        }
    }

    double avgR = totR / (double)N;
    double avgG = totG / (double)N;
    double avgB = totB / (double)N;

    double mad = 0.0;
    for (int i = y; i < y + height && i < img.size(); i++) {

```

```

        for (int j = x; j < x + width && j < img[0].size(); j++) {
            mad += abs(img[i][j][0] - avgR);
            mad += abs(img[i][j][1] - avgG);
            mad += abs(img[i][j][2] - avgB);
        }
    }

    return mad / N;
}

// Hitung max diff dalam blok RGB
double calMaxDiff(const vector<vector<RGB>>& img, 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 < img.size(); i++) {
        for (int j = x; j < x + width && j < img[0].size(); j++) {
            minR = min(minR, img[i][j][0]);
            minG = min(minG, img[i][j][1]);
            minB = min(minB, img[i][j][2]);
            maxR = max(maxR, img[i][j][0]);
            maxG = max(maxG, img[i][j][1]);
            maxB = max(maxB, img[i][j][2]);
        }
    }

    return (maxR - minR + maxG - minG + maxB - minB) / 3.0;
}

// Hitung entropy dalam blok RGB
double calEntropy(const vector<vector<RGB>>& img, int x, int y, int width, int height) {
    map<int, int> histogramR, histogramG, histogramB;
    int N = width * height;

    // Hitung histogram untuk setiap kanal warna
    for (int i = y; i < y + height && i < img.size(); i++) {
        for (int j = x; j < x + width && j < img[0].size(); j++) {
            histogramR[img[i][j][0]]++;
            histogramG[img[i][j][1]]++;
            histogramB[img[i][j][2]]++;
        }
    }

    // Hitung entropy untuk setiap kanal
    double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;

    for (const auto& pair : histogramR) {
        double p = (double)pair.second / N;

```

```

        if (p > 0) entropyR -= p * log2(p);
    }

    for (const auto& pair : histogramG) {
        double p = (double)pair.second / N;
        if (p > 0) entropyG -= p * log2(p);
    }

    for (const auto& pair : histogramB) {
        double p = (double)pair.second / N;
        if (p > 0) entropyB -= p * log2(p);
    }

    // Rata-rata entropy dari tiga kanal
    double entropy = (entropyR + entropyG + entropyB) / 3.0;

    return entropy;
}

double calSSIM(const vector<vector<RGB>>& imgRef, const vector<vector<RGB>>& imgPred, int x,
int y, int width, int height) {
    if (imgRef.empty() || imgRef[0].empty()) {
        std::cout << "imgRef kosong! SSIM tidak bisa dihitung." << std::endl;
        return 0.0;
    }

    const double C1 = (0.01 * 255) * (0.01 * 255);
    const double C2 = (0.03 * 255) * (0.03 * 255);
    const double wR = 0.2989, wG = 0.5870, wB = 0.1140;
    int N = width * height;

    std::vector<double> sumRef(3, 0.0), sumPred(3, 0.0);
    std::vector<double> varRef(3, 0.0), varPred(3, 0.0), cov(3, 0.0);

    for (int i = 0; i < height && y + i < imgRef.size(); ++i) {
        for (int j = 0; j < width && x + j < imgRef[0].size(); ++j) {
            const RGB& ref = imgRef[y + i][x + j];
            const RGB& pred = imgPred[i][j];

            for (int c = 0; c < 3; ++c) {
                sumRef[c] += ref[c];
                sumPred[c] += pred[c];
            }
        }
    }

    std::vector<double> meanRef(3), meanPred(3);
    for (int c = 0; c < 3; ++c) {

```

```

        meanRef[c] = sumRef[c] / N;
        meanPred[c] = sumPred[c] / N;
    }

    for (int i = 0; i < height && y + i < imgRef.size(); ++i) {
        for (int j = 0; j < width && x + j < imgRef[0].size(); ++j) {
            const RGB& ref = imgRef[y + i][x + j];
            const RGB& pred = imgPred[i][j];

            for (int c = 0; c < 3; ++c) {
                double diffRef = ref[c] - meanRef[c];
                double diffPred = pred[c] - meanPred[c];

                varRef[c] += diffRef * diffRef;
                varPred[c] += diffPred * diffPred;
                cov[c] += diffRef * diffPred;
            }
        }
    }

    for (int c = 0; c < 3; ++c) {
        varRef[c] /= (N - 1);
        varPred[c] /= (N - 1);
        cov[c] /= (N - 1);
    }

    double ssim[3];
    for (int c = 0; c < 3; ++c) {
        ssim[c] = ((2 * meanRef[c] * meanPred[c] + C1) * (2 * cov[c] + C2)) /
                  ((meanRef[c] * meanRef[c] + meanPred[c] * meanPred[c] + C1) * (varRef[c] +
varPred[c] + C2));
    }

    return wR * ssim[0] + wG * ssim[1] + wB * ssim[2];
}

```

inout.h

```

#ifndef INOUT_H
#define INOUT_H

#include <vector>
#include <array>
#include <string>

using RGB = std::array<int, 3>;

struct Node {

```

```

    int x, y;
    int width, height;
    RGB avgColor;
    bool isLeaf;
    Node* children[4];
};

std::vector<std::vector<RGB>> loadImage(const std::string& filename, int& width, int& height);

void fillImage(std::vector<std::vector<RGB>> &img, const Node* node);

void saveImage(const std::vector<std::vector<RGB>>& img, const std::string& filename);

void fillImageWithDepthLimit(std::vector<std::vector<RGB>>& image, Node* node, int maxDepth,
int currentDepth = 0);

void generateGif(const std::string& frameDir, const std::string& gifPath);

#endif

```

inout.cpp

```

#include <iostream>
#include <map>
#include <cmath>
#include <fstream>
#include <queue>
#include <sstream>
#include <FreeImage.h>
#include "headers/inout.h"

using namespace std;

vector<vector<RGB>> loadImage(const string& filename, int& width, int& height) {

    FreeImage_Initialise();
    FREE_IMAGE_FORMAT fif = FreeImage_GetFileType(filename.c_str(), 0);
    if (fif == FIF_UNKNOWN) {
        fif = FreeImage_GetFIFFFromFilename(filename.c_str());
    }

    if (fif == FIF_UNKNOWN || !FreeImage_FIFSupportsReading(fif)) {
        cerr << "Format file tidak didukung atau tidak bisa dibaca.\n";
        exit(1);
    }

    FIBITMAP* bitmap = FreeImage_Load(fif, filename.c_str());
    FIBITMAP* rgb_bitmap = FreeImage_ConvertTo24Bits(bitmap);

```

```

width = FreeImage_GetWidth(rgb_bitmap);
height = FreeImage_GetHeight(rgb_bitmap);

vector<vector<RGB>> image(height, vector<RGB>(width));

for (int y = 0; y < height; ++y) { // baris
    BYTE* bits = FreeImage_GetScanLine(rgb_bitmap, height - 1 - y); // FreeImage
    defaultnya terbalik (pertama di-scan = terakhir)
    for (int x = 0; x < width; ++x) { // kolom
        int b = bits[x * 3 + 0]; // FreeImage defaultnya BGR
        int g = bits[x * 3 + 1];
        int r = bits[x * 3 + 2];
        image[y][x] = { r, g, b }; // RGB (disesuaikan)
    }
}

FreeImage_Unload(rgb_bitmap);
FreeImage_Unload(bitmap);

FreeImage_DeInitialise();

return image;
}

void fillImage(vector<vector<RGB>> &img, const Node* node) {
    if (node == nullptr) return;

    int x = node->x, y = node->y, width = node->width, height = node->height;

    if (node->isLeaf) {
        RGB color = node->avgColor;
        for (int i = y; i < y + height; i++) {
            for (int j = x; j < x + width; j++) {
                if (i < img.size() && j < img[0].size()) {
                    img[i][j] = color;
                }
            }
        }
    } else {
        for (int i = 0; i < 4; i++) {
            if (node->children[i] != nullptr) {
                fillImage(img, node->children[i]);
            }
        }
    }
}

```

```

void saveImage(const vector<vector<RGB>>& img, const std::string& filename) {
    int width = img[0].size();
    int height = img.size();

    // Buat bitmap kosong
    FIBITMAP* bitmap = FreeImage_Allocate(width, height, 24);

    // Masukkan data piksel ke bitmap
    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            RGBQUAD color;
            color.rgbRed = img[height - 1 - y][x][0]; // Pembalikan posisi y
            color.rgbGreen = img[height - 1 - y][x][1]; // Pembalikan posisi y
            color.rgbBlue = img[height - 1 - y][x][2]; // Pembalikan posisi y

            FreeImage_SetPixelColor(bitmap, x, y, &color);
        }
    }

    // Simpan gambar ke file
    FreeImage_Save(FIF_JPEG, bitmap, filename.c_str(), JPEG_QUALITYGOOD);

    // Hapus bitmap dari memori
    FreeImage_Unload(bitmap);
}

void fillImageWithDepthLimit(std::vector<std::vector<RGB>>& image, Node* node, int maxDepth,
int currentDepth) {
    if (node == nullptr) return;

    if (node->isLeaf || currentDepth >= maxDepth) {
        RGB color = node->avgColor;
        for (int y = 0; y < node->height; y++) {
            for (int x = 0; x < node->width; x++) {
                if (node->y + y < image.size() && node->x + x < image[0].size()) {
                    image[node->y + y][node->x + x] = color;
                }
            }
        }
    } else {
        for (int i = 0; i < 4; i++) {
            if (node->children[i] != nullptr) {
                fillImageWithDepthLimit(image, node->children[i], maxDepth, currentDepth + 1);
            }
        }
    }
}

```

```

void generateGif(const std::string& frameDir, const std::string& gifPath) {

    std::string cmd = "magick -delay 50 -loop 0 " + frameDir + "/step_*.png " + gifPath;
    int result = system(cmd.c_str());
    if (result != 0) {
        std::cerr << "Error. Make sure ImageMagick is installed." << std::endl;
    }
}

```

quadtree.h

```

#ifndef QUADTREE_H
#define QUADTREE_H

#include <vector>
#include <array>
#include <string>
#include "inout.h"

Node* createNode(int x, int y, int width, int height);

void buildQuadtree(const std::vector<std::vector<RGB>>& image, Node* node, double threshold,
int min_size,
int errorMethod, bool createGIF);

int getDepth(Node* node);

int getNodeCount(Node* node);

RGB avgColor(const std::vector<std::vector<RGB>>& img, int x, int y, int width, int size);

int countNodes(Node* node);

void generateGifFrames(const std::vector<std::vector<RGB>>& originalImage, Node* root, const
std::string& frameDir);
#endif

```

quadtree.cpp

```

#include <iostream>
#include <map>
#include <cmath>
#include <fstream>
#include <queue>
#include <FreeImage.h>
#include <sstream>
#include <iomanip>

```

```

#include "headers/quadtreenode.h"
#include "headers/errorcounter.h"
#include <functional>

using namespace std;

Node* createNode(int x, int y, int width, int height) {
    Node* node = new Node;
    node->x = x;
    node->y = y;
    node->width = width;
    node->height = height;
    node->isLeaf = true;

    // Inisialisasi pointer anak dengan nullptr
    for (int i = 0; i < 4; ++i) {
        node->children[i] = nullptr;
    }

    return node;
}

// Hitung rata-rata RGB dari node leaf buat normalisasi
RGB avgColor(const vector<vector<RGB>>& img, int x, int y, int width, int height) {
    double sum[3] = {0};
    int N = width * height;

    for (int i = y; i < y + height && i < img.size(); ++i) {
        for (int j = x; j < x + width && j < img[0].size(); ++j) {
            for (int c = 0; c < 3; ++c) {
                sum[c] += img[i][j][c];
            }
        }
    }

    RGB avg = {0, 0, 0};
    if (N > 0) {
        for (int c = 0; c < 3; ++c) {
            avg[c] = sum[c] / N;
        }
    }
    return avg;
}

// Bangun Quadtree dan simpan ke array of Node
void buildQuadtreenode(const vector<vector<RGB>>& image, Node* node, double threshold, int min_size,
int errorMethod, bool createGIF) {

```

```

if (node == nullptr) return;

int x = node->x;
int y = node->y;
int width = node->width;
int height = node->height;
node->avgColor = avgColor(image, x, y, width, height);

double var;
if (errorMethod == 1) {
    var = calVariance(image, x, y, width, height);
} else if (errorMethod == 2) {
    var = calMAD(image, x, y, width, height);
} else if (errorMethod == 3) {
    var = calMaxDiff(image, x, y, width, height);
} else if (errorMethod == 4) {
    var = calEntropy(image, x, y, width, height);
} else if (errorMethod == 5) {
    vector<vector<RGB>> avgBlock(height, vector<RGB>(width, node->avgColor));
    var = 1.0 - calSSIM(image, avgBlock, x, y, width, height);
} else {
    cerr << "Error: Metode " << errorMethod << " tidak dikenal!" << endl;
    return;
}

bool isLeaf = (var <= threshold || width <= min_size || height <= min_size);
node->isLeaf = isLeaf;

if (!isLeaf) {
    int halfWidth = width / 2;
    int halfHeight = height / 2;

    if (halfWidth*halfHeight < min_size) {
        node->isLeaf = true;
        return;
    }

    node->children[0] = createNode(x, y, halfWidth, halfHeight); // TL
    node->children[1] = createNode(x + halfWidth, y, width - halfWidth, halfHeight); // TR
    node->children[2] = createNode(x, y + halfHeight, halfWidth, height - halfHeight); // BL
    node->children[3] = createNode(x + halfWidth, y + halfHeight, width - halfWidth, height - halfHeight); // BR

    for (int i = 0; i < 4; ++i) {
        buildQuadtree(image, node->children[i], threshold, min_size, errorMethod,
createGIF);
    }
}

```

```

    }

}

int getDepth(Node* node) {
    if (node == nullptr) return 0;
    if (node->isLeaf) return 1;

    int maxDepth = 0;
    for (int i = 0; i < 4; ++i) {
        maxDepth = max(maxDepth, getDepth(node->children[i]));
    }
    return 1 + maxDepth;
}

int countNodes(Node* node) {
    if (!node) return 0;
    int count = 1;
    for (int i = 0; i < 4; ++i) {
        count += countNodes(node->children[i]);
    }
    return count;
}

void generateGifFrames(const std::vector<std::vector<RGB>>& originalImage, Node* root, const std::string& frameDir) {

    int maxDepth = getDepth(root);

    for (int depth = 0; depth <= maxDepth; depth++) {

        std::vector<std::vector<RGB>> frame(originalImage.size(),
        std::vector<RGB>(originalImage[0].size()));

        fillImageWithDepthLimit(frame, root, depth);

        std::stringstream ss;
        ss << frameDir << "/step_" << std::setw(4) << std::setfill('0') << depth << ".png";
        saveImage(frame, ss.str());
    }
}

```

validation.h

```

#ifndef VALIDATION_H
#define VALIDATION_H

#include <string>

```

```

// Mengecek apakah path adalah absolut
bool isAbsolutePath(const std::string& path);

// Mengecek ekstensi file
bool isValidImageExtension(const std::string& path);
bool isValidGifExtension(const std::string& path);

// Mengecek apakah file eksis
bool fileExists(const std::string& path);

// Fungsi-fungsi input validasi
std::string getValidatedInputPath();
int getValidatedMethod();
int getValidatedMinSize();
std::string getValidatedOutputImagePath();
std::string getValidatedGifPath();
double getValidatedThreshold(int method);

#endif

```

validation.cpp

```

#include "headers/validation.h"
#include <iostream>
#include <fstream>
#include <set>
#include <algorithm>

using namespace std;

bool isAbsolutePath(const string& path) {
#ifdef _WIN32
    return path.length() > 2 && path[1] == ':' && (path[2] == '\\\\' || path[2] == '/');
#else
    return !path.empty() && path[0] == '/';
#endif
}

string getExtension(const string& path) {
    size_t dotPos = path.find_last_of('.');
    if (dotPos == string::npos) return "";
    string ext = path.substr(dotPos);
    // ubah ke huruf kecil semua
    transform(ext.begin(), ext.end(), ext.begin(), ::tolower);
    return ext;
}

```

```

bool isValidImageExtension(const string& path) {
    set<string> validExt = { ".png", ".jpg", ".jpeg", ".bmp" };
    return validExt.count(getExtension(path)) > 0;
}

bool isValidGifExtension(const string& path) {
    return getExtension(path) == ".gif";
}

bool fileExists(const string& path) {
    ifstream f(path.c_str());
    return f.good();
}

string getValidatedInputPath() {
    string path;
    while (true) {
        cout << "Masukkan path gambar input absolut: ";
        cin >> path;
        if (!isAbsolutePath(path)) {
            cout << "Path harus absolut.\n";
            continue;
        }
        if (!fileExists(path)) {
            cout << "File tidak ditemukan.\n";
            continue;
        }
        if (!isValidImageExtension(path)) {
            cout << "Jenis file tidak valid untuk gambar input.\n";
            continue;
        }
        return path;
    }
}

int getValidatedMethod() {
    int method;
    while (true) {
        cout << "Pilih metode error (1. Variance, 2. MAD, 3. MaxDiff, 4. Entropy, 5. SSIM): ";
        cin >> method;
        if (cin.fail() || method < 1 || method > 5) {
            cin.clear(); // Reset error flag
            cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Buang input yang salah
            cout << "Pilihan tidak valid. Harap masukkan angka antara 1 hingga 5.\n";
        } else {
            return method;
        }
    }
}

```

```

}

int getValidatedMinSize() {
    int minSize;
    while (true) {
        cout << "Masukkan ukuran blok minimum (positif): ";
        cin >> minSize;
        if (cin.fail() || minSize <= 0) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Ukuran blok minimum harus berupa bilangan bulat positif.\n";
        } else {
            return minSize;
        }
    }
}

string getValidatedOutputImagePath() {
    string path;
    while (true) {
        cout << "Masukkan path output gambar hasil: ";
        cin >> path;
        if (!isAbsolutePath(path)) {
            cout << "Path harus absolut.\n";
            continue;
        }
        if (!isValidImageExtension(path)) {
            cout << "Jenis file tidak valid untuk gambar output.\n";
            continue;
        }
        return path;
    }
}

string getValidatedGifPath() {
    string path;
    cout << "Masukkan path output GIF proses (kosongkan jika tidak ingin GIF): ";
    cin.ignore();
    getline(cin, path);
    if (path.empty()) return path;

    while (!isAbsolutePath(path) || !isValidGifExtension(path)) {
        if (!isAbsolutePath(path)) {
            cout << "Path harus absolut.\n";
        } else if (!isValidGifExtension(path)) {
            cout << "Ekstensi GIF harus .gif\n";
        }
    }
}

```

```

        cout << "Masukkan ulang path output GIF (atau kosongkan): ";
        getline(cin, path);
        if (path.empty()) break;
    }

    return path;
}

double getValidatedThreshold(int method) {
    double threshold;
    double minVal = 0, maxVal = 1;

    switch (method) {
        case 1: maxVal = 65025; break;
        case 2:
        case 3: maxVal = 255; break;
        case 4: maxVal = 8; break;
        case 5: maxVal = 1; break;
        default:
            cout << "Metode tidak valid. Default range digunakan (0-1).\n";
    }

    while (true) {
        cout << "Masukkan threshold (" << minVal << " - " << maxVal << "): ";
        cin >> threshold;
        if (threshold < minVal || threshold > maxVal) {
            cout << "Threshold di luar range.\n";
        } else {
            return threshold;
        }
    }
}

```

main.cpp

```

#include <iostream>
#include <map>
#include <cmath>
#include <fstream>
#include <queue>
#include <sstream>
#include <vector>
#include <chrono>
#include <FreeImage.h>
#include "headers/inout.h"
#include "headers/quadtree.h"
#include "headers/errorcounter.h"
#include "headers/validation.h"

```

```

using namespace std;

// Fungsi untuk menghitung ukuran file dalam byte
long long getFileSize(const string& filename) {
    ifstream file(filename, ios::binary | ios::ate);
    return file.is_open() ? static_cast<long long>(file.tellg()) : -1; // -1 jika gagal membaca file
}

int main() {
    string image_path = getValidatedInputPath();

    cout << "1. Variance: 0 - 65025 \n"
        << "2. MAD: 0 - 255 \n"
        << "3. MaxDiff: 0 - 255 \n"
        << "4. Entropy: 0 - 8 \n"
        << "5. SSIM: 0 - 1 \n";

    int method = getValidatedMethod();
    double threshold = getValidatedThreshold(method);
    int min_size = getValidatedMinSize();

    string output_image_path = getValidatedOutputImagePath();
    string output_gif_path = getValidatedGifPath();

    // Muat gambar
    int width, height;
    auto image = loadImage(image_path, width, height);
    if (image.empty()) {
        cerr << "Gagal memuat gambar!" << endl;
        return 1;
    }

    cout << "Gambar dimuat: " << width << "x" << height << endl;

    auto start = chrono::high_resolution_clock::now();

    Node* root = createNode(0, 0, width, height);
    string stepFrameDir = "steps";
    bool gifMode = !output_gif_path.empty();

    buildQuadtree(image, root, threshold, min_size, method, false);

    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> duration = end - start;

    vector<vector<RGB>> reconstructedImage(height, vector<RGB>(width));
}

```

```

fillImage(reconstructedImage, root);
saveImage(reconstructedImage, output_image_path);

cout << "Waktu eksekusi: " << duration.count() << " detik\n";

auto sizeBefore = getFileSize(image_path);
auto sizeAfter = getFileSize(output_image_path);
double compressionRatio = 1.0 - (double)sizeAfter / sizeBefore;

cout << "Ukuran gambar sebelum: " << sizeBefore << " bytes\n";
cout << "Ukuran gambar setelah: " << sizeAfter << " bytes\n";
cout << "Persentase kompresi: " << (compressionRatio * 100) << "%\n";

int depth = getDepth(root);
int nodeCount = countNodes(root);

cout << "Kedalaman pohon: " << depth << endl;
cout << "Banyak simpul: " << nodeCount << endl;

if (gifMode) {
    system(("rmdir /S /Q " + stepFrameDir).c_str());
    system(("mkdir " + stepFrameDir).c_str());

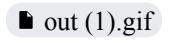
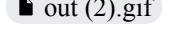
    generateGifFrames(image, root, stepFrameDir);
    generateGif(stepFrameDir, output_gif_path);
    cout << "GIF proses kompresi disimpan ke " << output_gif_path << endl;
}

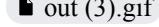
return 0;
}

```

BAB 3

TEST CASE

Nomor	Output statistik	Gambar input	Gambar hasil kompresi	GIF proses kompresi
1	<pre>Masukan path gambar input absolut: /Users/aliyahusnafayyaz/Downloads/kor1.jpg 1: Variance: 8 - 6983 2: MAD: 8 - 255 3: Entropy: 8 - 20 4: Settings: 8 - 8 5: Threshold: 1,2,3,A,B,C Masukan Output: kor1_out.jpg Masukan path output: /Users/aliyahusnafayyaz/Desktop/test/output/out.jpg Masukan path output GIF proses (karena jika tidak begini Gif2): /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Gambar dimut: 1352x985, 256 colors Ukuran gambar sebelum: 18834 bytes Ukuran gambar setelah: 21457 bytes Percentase kompresi: 8.8% Dokumen simpan: kor1_out.jpg Lip proses kompresi dilakukan ke /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Lip proses kompresi dilakukan ke /Users/aliyahusnafayyaz/Desktop/test/output/out.gif</pre>			
2	<pre>Masukan path gambar input absolut: /Users/aliyahusnafayyaz/Desktop/kor2.jpg 1: Variance: 8 - 6983 2: MAD: 8 - 255 3: Entropy: 8 - 20 4: Settings: 8 - 8 5: Threshold: 1,2,3,A,B,C Masukan Output: kor2_out.jpg Masukan path output: /Users/aliyahusnafayyaz/Desktop/test/output/out.jpg Masukan path output GIF proses (karena jika tidak begini Gif2): /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Gambar dimut: 1352x985, 256 colors Ukuran gambar sebelum: 18834 bytes Ukuran gambar setelah: 21457 bytes Percentase kompresi: 8.8% Dokumen simpan: kor2_out.jpg Lip proses kompresi dilakukan ke /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Lip proses kompresi dilakukan ke /Users/aliyahusnafayyaz/Desktop/test/output/out.gif</pre>			
3	<pre>Masukan path gambar input absolut: /Users/aliyahusnafayyaz/Desktop/kor2.jpg 1: Variance: 8 - 6983 2: MAD: 8 - 255 3: Entropy: 8 - 20 4: Settings: 8 - 8 5: Threshold: 1,2,3,A,B,C Masukan Output: kor2_out.jpg Masukan path output: /Users/aliyahusnafayyaz/Desktop/test/output/out.jpg Masukan path output GIF proses (karena jika tidak begini Gif2): /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Gambar dimut: 1352x985, 256 colors Ukuran gambar sebelum: 18834 bytes Ukuran gambar setelah: 21457 bytes Percentase kompresi: 8.8% Dokumen simpan: kor2_out.jpg Barang simpan: 37377</pre>			-
4	<pre>Masukan path gambar input absolut: /Users/aliyahusnafayyaz/Desktop/kor2.jpg 1: Variance: 8 - 6983 2: MAD: 8 - 255 3: Entropy: 8 - 20 4: Settings: 8 - 8 5: Threshold: 1,2,3,A,B,C Masukan Output: kor2_out.jpg Masukan path output: /Users/aliyahusnafayyaz/Desktop/test/output/out.jpg Masukan path output GIF proses (karena jika tidak begini Gif2): /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Gambar dimut: 1352x985, 256 colors Ukuran gambar sebelum: 18834 bytes Ukuran gambar setelah: 21457 bytes Percentase kompresi: 8.8% Dokumen simpan: kor2_out.jpg Barang simpan: 37377</pre>			-
5	<pre>Masukan path gambar input absolut: /Users/aliyahusnafayyaz/Desktop/male.png 1: Variance: 8 - 204 2: MAD: 8 - 204 3: Entropy: 8 - 20 4: Settings: 8 - 8 5: Threshold: 1,2,3,A,B,C Masukan Output: male_out.jpg Masukan path output: /Users/aliyahusnafayyaz/Desktop/test/output/out.jpg Masukan path output GIF proses (karena jika tidak begini Gif2): /Users/aliyahusnafayyaz/Desktop/test/output/out.gif Gambar dimut: 1352x985, 256 colors Ukuran gambar sebelum: 20886 bytes Ukuran gambar setelah: 20886 bytes Percentase kompresi: 100.0% Dokumen simpan: male_out.jpg Barang simpan: 32581</pre>			 *gambar png awal transparan

6	<pre>Masukan path gambar: /Users/alyhusna/fayzaa/Downloads/dice.png 1. Variance: 8 - 65025 2. Mean: 8 - 255 3. Median: 8 - 255 4. Entropy: 8 - 8 5. SVD: 8 - 8 Pilih metode error: {1,2,3,4,5}! 3 Masukan ukuran blok simbol: 35 Masukan ukuran blok simbol: 4 Masukan ukuran simbol pada file: 1 Masukan ukuran simbol pada file: 1 Gambar dimuat: 640x640 Ukuran gambar sebenar: 32737 bytes Ukuran gambar sebenar: 32737 bytes Percentase kompresi: 85.298% Karakter yang diidentifikasi: 1234567890 Banyak simbol: 6385</pre>			
7	<pre>Masukan path gambar: /Users/alyhusna/fayzaa/Downloads/testt.jpg 1. Variance: 8 - 65025 2. Mean: 8 - 255 3. Median: 8 - 255 4. Entropy: 8 - 8 5. SVD: 8 - 8 Pilih metode error: {1,2,3,4,5}! 4 Masukan ukuran blok simbol: 4 Masukan ukuran simbol pada file: 1 Gambar dimuat: 640x640 Ukuran gambar sebenar: 32737 bytes Ukuran gambar sebenar: 32737 bytes Percentase kompresi: 85.298% Karakter yang diidentifikasi: 1234567890 Banyak simbol: 6385</pre>			
8	<pre>alyhusna@Fayzaa-MacBook-Air:~/src\$./main Masukan path gambar: /Users/alyhusna/fayzaa/Downloads/orom1.jpg 1. Variance: 8 - 65025 2. Mean: 8 - 255 3. Median: 8 - 255 4. Entropy: 8 - 8 5. SVD: 8 - 8 Pilih metode error: {1,2,3,4,5}! 5 Masukan ukuran blok simbol: 4 Masukan ukuran simbol pada file: 1 Gambar dimuat: 640x640 Ukuran gambar sebenar: 32737 bytes Ukuran gambar sebenar: 32737 bytes Percentase kompresi: 85.298% Karakter yang diidentifikasi: 1234567890 Banyak simbol: 5301 GIF photo kompresi disimpan ke /Users/alyhusna/fayzaa/Tc112_13523055_13523062/test/output/out.gif</pre>			

BAB 4

HASIL DAN ANALISIS

4.1 Hasil dan Analisis

Percobaan kompresi gambar menggunakan algoritma *divide and conquer* yang berbasis Quadtree ini menunjukkan efektivitas pendekatan ini dalam menyederhanakan representasi gambar dengan tetap mempertahankan karakteristik spasial gambar tersebut. Gambar input akan diproses secara rekursif untuk dibagi menjadi blok-blok yang lebih kecil, sampai pada titik di mana tiap blok dapat direpresentasikan dengan satu nilai warna rata-rata yang masih memenuhi batas toleransi kesalahan (*threshold*). Pemrosesan ini dilakukan secara *top-down* dan bersifat hierarkis, sehingga hasil akhirnya berupa quadtree yang mewakili seluruh bagian dalam gambar.

Dari hasil uji coba dengan berbagai nilai *threshold*, ukuran blok minimum (*min_size*), dan metode perhitungan error, diperoleh bahwa akurasi hasil kompresi sangat bergantung pada pemilihan parameter-parameter tersebut. Semakin rendah nilai *threshold* yang digunakan, semakin besar struktur pohon yang dihasilkan karena lebih banyak blok yang harus dipecah demi memenuhi kriteria dalam presisi warna. Sebaliknya, jika *threshold* tinggi, blok-blok besar dapat langsung dianggap memenuhi tanpa perlu pembagian lanjutan, menghasilkan pohon dengan kedalaman yang lebih dangkal dan ukuran data yang lebih kecil, meskipun dengan penurunan kualitas gambar.

Dari sisi metode error, percobaan menunjukkan bahwa metode seperti *variance* dan *entropy* lebih sensitif terhadap perbedaan warna dalam gambar, yang mana menghasilkan lebih banyak pembagian dan hasil visual yang halus namun dengan struktur pohon yang lebih kompleks. Sementara itu, metode seperti *max diff* dan juga *MAD* bisa menghasilkan kompresi yang lebih cepat dan struktur pohon yang lebih kecil, namun cenderung menyatukan area dengan kontras yang tinggi sehingga dapat mengurangi kualitas hasil gambar. Selain itu, ada metode SSIM mempertimbangkan persepsi manusia terhadap kualitas sehingga memberikan hasil terbaik secara visual, namun membutuhkan komputasi tambahan membandingkan terhadap gambar asli.

4.2 Kompleksitas Algoritma

Dari sisi kompleksitas algoritma ini, strategi *divide and conquer* yang diterapkan dalam program ini bersifat rekursif dan memecah area gambar menjadi empat sub-area pada setiap langkah pembagian, oleh karena itu, analisis kompleksitas dapat dilihat dari tiga sisi, yaitu :

- **Kasus terbaik (*best case*)** : Terjadi ketika seluruh blok gambar langsung memenuhi syarat sebagai simpul daun tanpa perlu dibagi lagi. Ini bisa terjadi jika *threshold* yang digunakan sangat tinggi atau gambar sangat homogen. Dalam kondisi ini, algoritma hanya perlu melakukan sedikit rekursi, sehingga kompleksitas waktunya mendekati **O(1)** untuk setiap simpul, dan totalnya menjadi **O(n)** di mana **n** adalah jumlah blok utama.
- Kasus rata-rata (average case) : Terjadi ketika beberapa blok memenuhi syarat sebagai daun dan sebagian lainnya perlu dibagi. Pada skenario seperti ini, kedalaman pohon bervariasi dan jumlah simpul bertambah secara logaritmik terhadap ukuran gambar. Karena setiap pembagian memperkecil ukuran blok menjadi setengah secara horizontal dan vertikal, maka total jumlah simpul dalam banyak kasus berada dalam orde **O(n log n)**.

- Kasus terburuk (worst case) : Terjadi ketika semua blok perlu dibagi hingga mencapai batas minimum ukuran blok. Ini akan menghasilkan sebuah pohon yang sangat dalam dan jumlah simpul yang besar. Untuk gambar berukuran $w \times h$ piksel, dan ukuran minum blok $s \times s$, maka jumlah maksimum simpul adalah sekitar $(w/s) \times (h/s)$, menghasilkan kompleksitas waktu hingga mencapai $O(n^2)$ dengan n adalah jumlah piksel.

Dari segi kompleksitas ruang, algoritma ini memerlukan alokasi simpul bagi setiap blok yang diproses. Oleh karena itu, kompleksitas ruangnya berada di kisaran $O(k)$ dengan k adalah jumlah simpul dalam pohon. Ruang tambahan juga diperlukan untuk menyimpan informasi warna rata-rata, posisi, ukuran, serta pointer ke simpul.

4.3 Kesimpulan

Secara keseluruhan, implementasi algoritma *divide and conquer* dalam kompresi gambar melalui Quadtree ini memberikan hasil yang fleksibel dan cukup efisien, tergantung pada parameter yang digunakan. Algoritma ini sangat baik digunakan pada gambar yang memiliki struktur spasial yang jelas dan area homogen yang luas. Meskipun kompleksitas algoritma terburuknya bisa menjadi tinggi, dalam praktik sebenarnya, dengan pemilihan *threshold* dan *minimum size* yang tepat, performa akan tetap stabil dan menghasilkan struktur pohon yang jelas. Quadtree juga memiliki keuntungan tambahan berupa kemudahan dalam melakukan rekonstruksi gambar serta kemungkinan untuk dapat dioptimasi lebih lanjut pada level-level tertentu.

BAB 5

PENJELASAN ALGORITMA BONUS

5.1 Implementasi Structural Similarity Index Measure (SSIM)

SSIM (*Structural Similarity Index Measure*) adalah sebuah perhitungan metode error untuk mengukur persamaan antara dua gambar berdasarkan persepsi visual manusia. Tidak seperti metode lainnya yang hanya melihat perbedaan nilai numerik, SSIM mempertimbangkan tiga komponen utama, yaitu Luminasi, Kontras, dan Struktur. SSIM menghasilkan nilai antara 0 hingga 1, dengan 1 berarti gambar identik, dan semakin mendekati 0, maka semakin besar perbedaan visualnya. SSIM bekerja dengan membandingkan dua gambar blok demin blok, menghitung rata-rata, variansi, dan kovariansi dari piksel-piksel yang dibandingkan, kemudian menggabungkannya dalam sebuah rumus.

Ada beberapa tahapan dalam perhitungan dengan SSIM ini, yaitu:

1. Membuat representasi sederhana (*avgBlock*) : Sebelum menghitung SSIM, dalam fungsi *buildQuadtree*, dibuat sebuah blok baru bernama *avgBlock*. Blok ini berisi warna rata-rata (*avgColor*) dari blok gambar asli yang sedang diproses. Semua piksel di dalam *avgBlock* memiliki warna yang sama, sehingga mewakili versi *compressed* dari blok. Inilah yang akan dibandingkan dengan gambar asli menggunakan SSIM.
2. Fungsi *calSSIM* : Dalam fungsi *calSSIM* ini akan menghitung nilai SSIM antara blok asli (*imgRef*) dan blok hasil kompresi (*avgBlock / imgPred*). Ada beberapa proses, yaitu :
 - a. Menghitung rata-rata warna per channel (R, G, B)
 - b. Menghitung variansi dan kovariansi untuk mengetahui sebaran dan korelasi warna
 - c. Menggabungkan nilai-nilai tersebut ke dalam rumus SSIM
 - d. Menghasilkan skor SSIM untuk masing-masing channel, lalu dirata-ratakan menggunakan bobot : R (29.89%), G (58.7%), B (11.4%)F
3. Penerapan SSIM di fungsi *buildQuadtree*: Dalam fungsi *buildQuadtree*, SSIM digunakan untuk menentukan apakah suatu blok sudah cukup mirip dengan representasi rata-ratanya. Nilai error dihitung pada bagian `var = 1.0 - calSSIM()`. Jika error tersebut lebih kecil atau sama dengan *threshold*, maka blok dianggap cukup representatif dan tidak dipecah lagi (menjadi daun). Jika tidak, maka blok akan dibagi menjadi empat bagian dan proses akan diulang untuk masing-masing bagian.

5.2 Visualisasi Proses Pembentukan Quadtree dengan Format GIF

Untuk memfasilitasi pelaksanaan bonus ini, digunakan tiga fungsi yaitu *generateGifFrames*, *generateGif*, serta *fillImageWithDepth*. Berikut adalah penjelasannya,

1. Fungsi *generateGifFrames* digunakan untuk membuat urutan gambar/*frame* dari proses pembentukan quadtree. *Frame-frame* inilah yang nantinya akan dijadikan GIF visualisasi proses kompresi gambar. Pertama-tama, fungsi akan mencari kedalaman pohon untuk menentukan berapa banyak *frame* yang akan dibuat. Selanjutnya, fungsi akan melakukan iterasi dari kedalaman 0 sampai ke kedalaman maksimum. Awalnya dibuat *frame* kosong berbentuk matriks vektor RGB sebagai inisialisasi, lalu dipanggil fungsi *fillImageWithDepthLimit* yang akan

mengisi *frame* dengan warna sampai ke simpul yang berkaitan. Selanjutnya, *frame* akan disimpan sebagai file gambar.

2. Fungsi *fillImageWithDepthLimit* digunakan untuk mengisi gambar berdasarkan hasil *quadtree*, namun hanya sampai kedalaman tertentu. Cara kerja fungsi ini adalah jika simpul merupakan daun atau sudah merupakan simpul di kedalaman yang diminta, akan dibentuk matriks vektor RGB yang diisi dengan rata-rata warna RGB yang dimiliki simpul tersebut. Jika bukan, maka akan terus dilakukan rekursi sampai kondisi terpenuhi.
3. Fungsi *generateGif* digunakan untuk membuat file GIF dari *frame-frame* yang telah dibuat oleh *generateGifFrames*. Fungsi ini menggunakan perintah ImageMagick untuk menggabungkan gambar menjadi animasi. Dalam implementasinya, fungsi ini membentuk string perintah "`magick -delay 50 -loop 0 <frameDir>/step_*.png <gifPath>`" untuk menggabungkan seluruh frame menjadi satu file .gif

LAMPIRAN

Pranala ke *repository*: https://github.com/aliyahusnaf/Tucil2_13523055_13523062.git

Tabel keterselesaian:

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	