

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2024/2025

Disusun Oleh :
13523134 Sebastian Enrico Nathanael
13523140 Mahesa Fadhillah Andre

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

Bab I

Deskripsi Tugas

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

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



Gambar 2. Proses Pembentukan Quadtree dalam Kompresi Gambar

(Sumber:https://miro.medium.com/v2/resize:fit:640/format:webp/1*LHD7PsbmbgNBFrYkxyG5dA.gif)

Cek sumber untuk melihat animasi GIF

Ilustrasi kasus :

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- Metode perhitungan variansi: pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada Tabel 1.
- Threshold variansi: nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
- Minimum block size: ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai Tabel 1.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- ❖ Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- ❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

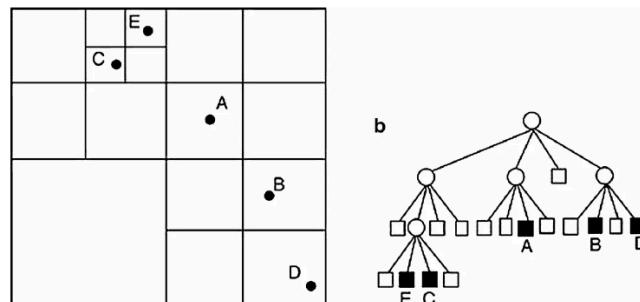
5. Rekursi dan Penghentian

Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- ❖ Error blok berada di bawah threshold.
- ❖ Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar

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

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar.

Jika error dalam blok melebihi ambas batas (threshold), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Bab II

Dasar Teori

2.1 Pengertian Algoritma Divide and Conquer

Algoritma Divide and Conquer (bagi dan takluk) merupakan salah satu paradigma desain algoritma yang digunakan untuk menyelesaikan masalah kompleks dengan cara memecahnya menjadi submasalah yang lebih kecil dan lebih mudah dikelola. Pendekatan ini bekerja dengan tiga langkah utama: pembagian (divide), penyelesaian (conquer), dan penggabungan (combine). Submasalah yang dihasilkan memiliki sifat yang serupa dengan masalah awal, tetapi dengan skala yang lebih kecil, sehingga dapat diselesaikan secara rekursif. Setelah submasalah terselesaikan, solusinya digabungkan untuk membentuk solusi dari masalah asli.

Pendekatan ini efektif untuk masalah yang dapat dipecah menjadi bagian-bagian independen, di mana setiap bagian dapat diselesaikan tanpa memengaruhi bagian lainnya. Divide and Conquer sering digunakan dalam pemrograman komputer karena kemampuannya untuk mengurangi kompleksitas waktu komputasi, terutama pada masalah yang melibatkan data dalam jumlah besar.

2.2 Langkah-langkah Algoritma Divide and Conquer

Secara umum, algoritma Divide and Conquer mengikuti tiga langkah berikut:

1. Divide (Pembagian): Masalah besar dipecah menjadi beberapa submasalah yang lebih kecil. Pembagian ini biasanya dilakukan secara berulang hingga submasalah menjadi cukup sederhana untuk diselesaikan langsung.
2. Conquer (Penyelesaian): Setiap submasalah diselesaikan secara rekursif. Jika submasalah telah mencapai kondisi dasar (basis rekursi), maka submasalah tersebut diselesaikan dengan metode langsung tanpa pembagian lebih lanjut.
3. Combine (Penggabungan): Solusi dari submasalah-submasalah yang telah diselesaikan digabungkan untuk membentuk solusi dari masalah awal. Langkah ini sering kali menjadi bagian kritis karena memerlukan strategi yang tepat untuk memastikan hasil akhir benar.

2.4 Analisis Kompleksitas

Kompleksitas waktu dari algoritma Divide and Conquer biasanya dianalisis menggunakan persamaan rekurensi. Secara umum, persamaan rekurensi untuk algoritma Divide and Conquer dapat dituliskan sebagai:

$$T(n) = a \cdot T\left(\frac{n}{b}\right) + f(n)$$

Di mana:

- $T(n)$: Total waktu untuk menyelesaikan masalah berukuran n .
- a : Jumlah submasalah yang dihasilkan pada setiap langkah pembagian.
- $\left(\frac{n}{b}\right)$: Ukuran setiap submasalah.
- $f(n)$: Waktu yang diperlukan untuk membagi masalah dan menggabungkan solusi.

Bab III

Kode Program

Struktur File

```
Tucil2_13523134_13523140/
├── bin/
│   └── program.exe
├── doc/
│   └── Laporan_Tucil2_13523134_13523140
└── src/
    ├── ErrorMeasurement/
    │   └── Entropy/
    │       └── Entropy.cpp
    ├── MAD/
    │   └── MAD.cpp
    └── MaxPixelDiff/
        └── MaxPixelDiff.cpp
```

```

|   |   └── Variance/
|   |       └── Variance.cpp
|   ├── ImageIO/
|   |   ├── ImageIO.cpp
|   |   ├── stb_image_write.h
|   |   └── stb_image.h
|   ├── Quadtree/
|   |   └── Quadtree.cpp
|   └── main.cpp
└── test/

```

File Entropy.cpp

```

#include <vector>
#include <cmath>
#include <iostream>
using namespace std;

// Fungsi untuk menghitung entropi dalam satu blok untuk setiap kanal RGB
double calculateBlockEntropy(const vector<vector<vector<unsigned char>>> &rgb, int startX, int startY, int width, int height)
{
    // Array untuk menghitung frekuensi setiap intensitas (0-255) untuk R, G, B
    vector<int> histR(256, 0), histG(256, 0), histB(256, 0);
    int totalPixels = 0;

    // hitung histogram untuk setiap rgb
    for (int y = startY; y < startY + height && y < (int)rgb.size(); y++)
    {
        for (int x = startX; x < startX + width && x < (int)rgb[y].size(); x++)
        {
            histR[rgb[y][x][0]]++; // R
            histG[rgb[y][x][1]]++; // G
            histB[rgb[y][x][2]]++; // B
            totalPixels++;
        }
    }

    if (totalPixels == 0)
    {
        return 0;
    }

    // hitung entropi untuk setiap kanal

```

```

double entropyR = 0, entropyG = 0, entropyB = 0;
for (int i = 0; i < 256; i++)
{
    if (histR[i] > 0)
    {
        double p = static_cast<double>(histR[i]) / totalPixels;
        entropyR -= p * log2(p);
    }
    if (histG[i] > 0)
    {
        double p = static_cast<double>(histG[i]) / totalPixels;
        entropyG -= p * log2(p);
    }
    if (histB[i] > 0)
    {
        double p = static_cast<double>(histB[i]) / totalPixels;
        entropyB -= p * log2(p);
    }
}

double entropy = (entropyR + entropyG + entropyB) / 3;
return entropy;
}

```

$$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$$

$$H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok

$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

Untuk Metode Entropy pertama-tama inisiasi `double entropyR = 0, entropyG = 0, entropyB = 0;`

File MAD.cpp

```

#include <iostream>
#include <vector>
#include <cmath>
using namespace std;

```

```

// menghitung Mean Absolute Deviation (MAD) antara dua matriks gambar rgb
double calculateBlockMAD(const vector<vector<vector<unsigned char>>> &rgb, int startX, int
startY, int width, int height)
{
    vector<double> valR, valG, valB;

    for (int y = startY; y < startY + (height) && y < (int)rgb.size(); y++)
    {
        for (int x = startX; x < startX + (width) && x < (int)rgb[y].size(); x++)
        {
            valR.push_back(rgb[y][x][0]); // R
            valG.push_back(rgb[y][x][1]); // G
            valB.push_back(rgb[y][x][2]); // B
        }
    }

    int N = height * width;

    if (N == 0)
    {
        return 0;
    }

    // hitung rata-rata rgb
    double avgR = 0, avgG = 0, avgB = 0;
    for (int i = 0; i < N; i++)
    {
        avgR += valR[i];
        avgG += valG[i];
        avgB += valB[i];
    }
    avgR /= N;
    avgG /= N;
    avgB /= N;

    // hitung mad untuk setiap rgb
    double madR = 0;
    double madG = 0;
    double madB = 0;
    for (int i = 0; i < N; i++)
    {
        madR += abs(valR[i] - avgR);
        madG += abs(valG[i] - avgG);
        madB += abs(valB[i] - avgB);
    }

    madR /= N;
    madG /= N;
    madB /= N;

    // hitung MAD rata-rata untuk rgb

```

```

        double MAD = (madR + madG + madB) / 3;
    return MAD;
}

```

File MaxPixelDiff.cpp

```

#include <vector>
#include <cmath>
#include <iostream>
using namespace std;

// menghitung Max Pixel Difference dalam satu blok untuk rgb
double calculateBlockMaxPixelDifference(const vector<vector<vector<unsigned char>>> &rgb, int startX, int startY, int width, int height)
{
    unsigned char maxR = 0, maxG = 0, maxB = 0;
    unsigned char minR = 255, minG = 255, minB = 255;
    bool DataExist = false;

    for (int y = startY; y < startY + height && y < (int)rgb.size(); y++)
    {
        for (int x = startX; x < startX + width && x < (int)rgb[y].size(); x++)
        {
            DataExist = true;
            maxR = max(maxR, rgb[y][x][0]); // R max
            maxG = max(maxG, rgb[y][x][1]); // G max
            maxB = max(maxB, rgb[y][x][2]); // B max
            minR = min(minR, rgb[y][x][0]); // R min
            minG = min(minG, rgb[y][x][1]); // G min
            minB = min(minB, rgb[y][x][2]); // B min
        }
    }

    if (!DataExist)
    {
        return 0;
    }

    // hitung Max Pixel Difference rgb
    double diffR = maxR - minR;
    double diffG = maxG - minG;
    double diffB = maxB - minB;

    // hitung Max Pixel Difference
    double maxPixelDiff = (diffR + diffG + diffB) / 3;
    return maxPixelDiff;
}

```

File Variance.cpp

```

#include <vector>
#include <string>
#include <cmath>
using namespace std;

// menghitung variansi dalam satu blok
double calculateBlockVariance(const vector<vector<vector<unsigned char>>> &rgb, int startX,
int startY, int width, int height)
{
    vector<double> valR, valG, valB;

    // masukkan nilai RGB dari blok ke vector val_rgb
    for (int y = startY; y < startY + (height) && y < (int)rgb.size(); y++)
    {
        for (int x = startX; x < startX + (width) && x < (int)rgb[y].size(); x++)
        {
            valR.push_back(rgb[y][x][0]); // R
            valG.push_back(rgb[y][x][1]); // G
            valB.push_back(rgb[y][x][2]); // B
        }
    }

    int N = height * width;

    if (N == 0)
    {
        return 0;
    }

    // hitung rata-rata rgb
    double avgR = 0, avgG = 0, avgB = 0;
    for (int i = 0; i < N; i++)
    {
        avgR += valR[i];
        avgG += valG[i];
        avgB += valB[i];
    }
    avgR /= N;
    avgG /= N;
    avgB /= N;

    // hitung variansi untuk setiap rgb
    double varR = 0;
    double varG = 0;
    double varB = 0;
    for (int i = 0; i < N; i++)
    {
        varR += (pow(valR[i] - avgR, 2));
        varG += (pow(valG[i] - avgG, 2));
        varB += (pow(valB[i] - avgB, 2));
    }
}

```

```

    varR /= N;
    varG /= N;
    varB /= N;

    // hitung variansi
    double variance = (varR + varG + varB) / 3;
    return variance;
}

```

File ImageIO.cpp

```

#include <iostream>
#include <vector>
#include <string>

#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image.h"
#include "stb_image_write.h"

using namespace std;

vector<vector<vector<unsigned char>>> loadImage(const string& path, int& width, int& height) {
    int channels;

    unsigned char* data = stbi_load(path.c_str(), &width, &height, &channels, 3);

    if (!data) {
        cerr << "Gagal load gambar: " << path << endl;
        cerr << "Alasan: " << stbi_failure_reason() << endl;
        exit(1);
    }

    vector<vector<vector<unsigned char>>> rgb(height, vector<vector<unsigned char>>(width,
vector<unsigned char>(3)));

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;
            rgb[y][x][0] = data[idx + 0];
            rgb[y][x][1] = data[idx + 1];
            rgb[y][x][2] = data[idx + 2];
        }
    }

    stbi_image_free(data);
    return rgb;
}

```

```

void saveImage(const string& path, const vector<vector<vector<unsigned char>>>& rgb, int width,
int height, string extension) {
    vector<unsigned char> data(width * height * 3);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;
            data[idx + 0] = rgb[y][x][0]; // R
            data[idx + 1] = rgb[y][x][1]; // G
            data[idx + 2] = rgb[y][x][2]; // B
        }
    }

    int success;

    if (extension == ".jpg") {
        success = stbi_write_jpg(path.c_str(), width, height, 3, data.data(), width * 3);
    }
    else if (extension == ".png") {
        success = stbi_write_png(path.c_str(), width, height, 3, data.data(), width * 3);
    }
    else if (extension == ".jpeg") {
        success = stbi_write_jpg(path.c_str(), width, height, 3, data.data(), width * 3);
    }

    if (!success) {
        cerr << "Gagal menyimpan gambar ke: " << path << endl;
    } else {
        cout << "Gambar berhasil disimpan ke: " << path << endl;
    }
}

```

File Quadtree.cpp

```

#include <iostream>
#include <vector>
#include <cmath>
#include <string>
#include "..\ErrorMeasurement\MAD\MAD.h"
#include "..\ErrorMeasurement\MaxPixelDiff\MaxPixelDiff.cpp"
#include "..\ErrorMeasurement\Variance\Variance.cpp"
#include "..\ErrorMeasurement\Entropy\Entropy.cpp"

using namespace std;

struct Node
{
    int x, y;
    int width, height;
    bool isLeaf;

```

```

        unsigned char avgR;
        unsigned char avgG;
        unsigned char avgB;

        Node *topLeft;
        Node *topRight;
        Node *bottomLeft;
        Node *bottomRight;
    };

void getAverageColor(
    const vector<vector<vector<unsigned char>>> &rgb,
    int x, int y, int width, int height,
    unsigned char &avgR, unsigned char &avgG, unsigned char &avgB)
{
    long long sumR = 0, sumG = 0, sumB = 0;

    int validWidth = min(width, max(0, (int)rgb[0].size() - x));
    int validHeight = min(height, max(0, (int)rgb.size() - y));

    if (validWidth <= 0 || validHeight <= 0 || x < 0 || y < 0)
    {
        avgR = avgG = avgB = 0;
        return;
    }

    for (int i = y; i < y + validHeight; i++)
    {
        for (int j = x; j < x + validWidth; j++)
        {
            sumR += rgb[i][j][0];
            sumG += rgb[i][j][1];
            sumB += rgb[i][j][2];
        }
    }

    int total = validWidth * validHeight;
    avgR = static_cast<unsigned char>(sumR / total);
    avgG = static_cast<unsigned char>(sumG / total);
    avgB = static_cast<unsigned char>(sumB / total);
}

Node *buildQuadtree(
    const vector<vector<vector<unsigned char>>> &rgb,
    int x, int y, int width, int height,
    int minBlockSize, double threshold, string errorMethod)
{
    Node *node = new Node();
    node->x = x;
    node->y = y;
    node->width = width;
}

```

```

node->height = height;

    getAverageColor(rgb, x, y, width, height, node->avgR, node->avgG, node->avgB);
if (width * height <= minBlockSize)
{
    node->isLeaf = true;
    node->topLeft = node->topRight = node->bottomLeft = node->bottomRight = nullptr;
    return node;
}

double variance = 0.0;

if (errorMethod == "Variance")
{
    variance = calculateBlockVariance(rgb, x, y, width, height);
}
else if (errorMethod == "MAD")
{
    variance = calculateBlockMAD(rgb, x, y, width, height);
}
else if (errorMethod == "MaxPixelDifference")
{
    variance = calculateBlockMaxPixelDifference(rgb, x, y, width, height);
}
else if (errorMethod == "Entropy")
{
    variance = calculateBlockEntropy(rgb, x, y, width, height);
}

if (variance <= threshold)
{
    node->isLeaf = true;
    node->topLeft = node->topRight = node->bottomLeft = node->bottomRight = nullptr;
    return node;
}

node->isLeaf = false;

int halfWidth = (width + 1) / 2;
int halfHeight = (height + 1) / 2;

node->topLeft = buildQuadtree(rgb, x, y, halfWidth, halfHeight, minBlockSize, threshold,
errorMethod);
node->topRight = buildQuadtree(rgb, x + halfWidth, y, width - halfWidth, halfHeight,
minBlockSize, threshold, errorMethod);
node->bottomLeft = buildQuadtree(rgb, x, y + halfHeight, halfWidth, height - halfHeight,
minBlockSize, threshold, errorMethod);
node->bottomRight = buildQuadtree(rgb, x + halfWidth, y + halfHeight, width - halfWidth,
height - halfHeight, minBlockSize, threshold, errorMethod);

return node;

```

```

}

void deleteTree(Node *root)
{
    if (!root)
        return;

    deleteTree(root->topLeft);
    deleteTree(root->topRight);
    deleteTree(root->bottomLeft);
    deleteTree(root->bottomRight);

    delete root;
}

void fillArea(
    vector<vector<vector<unsigned char>>> &output,
    int x, int y, int width, int height,
    unsigned char r, unsigned char g, unsigned b)
{
    for (int i = y; i < y + height; i++)
    {
        for (int j = x; j < x + width; j++)
        {
            output[i][j][0] = r;
            output[i][j][1] = g;
            output[i][j][2] = b;
        }
    }
}

void reconstructImage(
    Node *root,
    vector<vector<vector<unsigned char>>> &outputImage)
{
    if (!root)
        return;

    if (root->isLeaf)
    {
        fillArea(outputImage, root->x, root->y, root->width, root->height, root->avgR,
root->avgG, root->avgB);
    }
    else
    {
        reconstructImage(root->topLeft, outputImage);
        reconstructImage(root->topRight, outputImage);
        reconstructImage(root->bottomLeft, outputImage);
        reconstructImage(root->bottomRight, outputImage);
    }
}

```

```

// Fungsi untuk menghitung kedalaman pohon
int calculateTreeDepth(Node *root)
{
    if (!root)
        return 0;
    if (root->isLeaf)
        return 1;

    int depthTL = calculateTreeDepth(root->topLeft);
    int depthTR = calculateTreeDepth(root->topRight);
    int depthBL = calculateTreeDepth(root->bottomLeft);
    int depthBR = calculateTreeDepth(root->bottomRight);

    return 1 + max(max(depthTL, depthTR), max(depthBL, depthBR));
}

// Fungsi untuk menghitung jumlah simpul
int calculateNodeCount(Node *root)
{
    if (!root)
        return 0;
    if (root->isLeaf)
        return 1;

    return 1 + calculateNodeCount(root->topLeft) +
            calculateNodeCount(root->topRight) +
            calculateNodeCount(root->bottomLeft) +
            calculateNodeCount(root->bottomRight);
}

```

File main.cpp

```

#include <iostream>
#include <limits>
#include <string>
#include <vector>
#include <chrono>
#include <algorithm>
#include <cctype>
#include <filesystem>
#include "..\src\Quadtree\Quadtree.cpp"
#include "..\src\ImageIO\ImageIO.cpp"
namespace fs = std::filesystem;
using namespace std;

// input
// 1. input alamat file gambar input ()
// 2. input metode perhitungan error (done)
// 3. input ambang batas (threshold) (done)

```



```

if (!fs::exists(inputPath))
{
    cout << "Error: File tidak ditemukan. Silakan masukkan alamat file yang valid.\n";
    // Tambahkan informasi tambahan
    fs::path pathObj(inputPath);
    cout << "Direktori: " << pathObj.parent_path().string() << endl;
    cout << "Nama file: " << pathObj.filename().string() << endl;
    if (!fs::exists(pathObj.parent_path()))
    {
        cout << "Direktori tidak ditemukan!\n";
    }
    else
    {
        cout << "Direktori ditemukan, tetapi file tidak ada.\n";
    }
    continue;
}
fs::path pathObj(inputPath);
extension = pathObj.extension().string();
transform(extension.begin(), extension.end(), extension.begin(), ::tolower);
if (extension != ".jpg" && extension != ".jpeg" && extension != ".png")
{
    cout << "Error: File harus berekstensi .jpg, .jpeg, atau .png. Ekstensi yang dimasukkan: " << extension << endl;
    continue;
}
break;
}

while (true)
{
    cout << "Pilih metode perhitungan error" << endl;
    cout << "1. Variance" << endl;
    cout << "2. Mean Absolute Deviation (MAD)" << endl;
    cout << "3. Max Pixel Difference" << endl;
    cout << "4. Entropy" << endl;
    cout << "Pilih metode: ";
    cin >> pilihmetode;

    // error message handling pilihan metode
    if (cin.fail())
    {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Terdapat kesalahan: Masukan harus berupa angka. Silakan coba lagi (gunakan angka)." << endl;
        continue;
    }

    if (pilihmetode < 1 || pilihmetode > 4)
    {

```

```

        cout << "Kesalahan pada input: harap masukkan angka antara 1 - 4." << endl;
        continue;
    }
    break;
}
// input threshold
while (true)
{
    cout << "Masukkan nilai threshold: ";
    cin >> threshold;
    if (cin.fail())
    {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Terdapat kesalahan: Masukan harus berupa angka. Silakan coba lagi
(gunakan angka)." << endl;
        continue;
    }

    if (threshold < 0)
    {
        cout << "Kesalahan pada input: Threshold tidak boleh negatif. Silakan masukkan
nilai yang valid.\n";
        continue;
    }
    break;
}

// input ukuran blok minimum
while (true)
{
    cout << "Masukkan ukuran blok minimum: ";
    cin >> minBlockSize;
    if (cin.fail())
    {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Error: Masukan harus berupa angka. Silakan coba lagi.\n";
        continue;
    }

    if (minBlockSize < 0)
    {
        cout << "Error: Ukuran blok minimum tidak boleh negatif. Silakan masukkan nilai
yang valid.\n";
        continue;
    }
    break;
}

// input alamat output gambar

```

```

cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Bersihkan buffer
while (true)
{
    cout << "Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): ";
    getline(cin, outputPath);
    if (outputPath.empty())
    {
        cout << "Error: Alamat file tidak boleh kosong. Silakan coba lagi.\n";
        continue;
    }
    // Hapus tanda kutip di awal dan akhir jika ada
    if (outputPath.front() == '\"' && outputPath.back() == '\"')
    {
        outputPath = outputPath.substr(1, outputPath.length() - 2);
    }
    // Cek apakah direktori output valid
    fs::path outputPathObj = fs::path(outputPath);
    fs::path outputDir = outputPathObj.parent_path();
    if (!outputDir.empty() && !fs::exists(outputDir))
    {
        cout << "Error: Direktori output tidak ditemukan. Silakan masukkan alamat yang valid.\n";
        continue;
    }
    // Hapus tanda kutip di awal dan akhir jika ada
    if (outputPath.front() == '\"' && outputPath.back() == '\"')
    {
        outputPath = outputPath.substr(1, outputPath.length() - 2);
    }
    // Cek apakah outputPath adalah direktori
    if (fs::exists(outputPathObj) && fs::is_directory(outputPathObj))
    {
        // Jika direktori, tambahkan nama file default
        outputPath = outputPathObj.string() + "\\hasil.png";
        cout << "Peringatan: Anda memasukkan direktori. Output akan disimpan sebagai: " << outputPath << endl;
    }
    break;
}

// kompresi dimulai
auto start = chrono::high_resolution_clock::now(); // menghitung waktu

// Muat gambar
vector<vector<vector<unsigned char>>> rgb;
try
{
    rgb = loadImage(inputPath, width, height);
}
catch (const exception &e)
{

```

```

        cout << "Gagal memuat gambar: " << e.what() << endl;
        return 1;
    }
    vector<vector<vector<unsigned char>>> output(height, vector<vector<unsigned char>>(width,
vector<unsigned char>(3)));

    unsigned char avgR, avgG, avgB;

    string errorMethod;
    switch (pilihmetode)
    {
    case 1:
        errorMethod = "Variance";
        break;
    case 2:
        errorMethod = "MAD";
        break;
    case 3:
        errorMethod = "MaxPixelDifference";
        break;
    case 4:
        errorMethod = "Entropy";
        break;
    default:
        cout << "Kesalahan pada input: harap masukkan angka antara 1 - 4" << endl;
        return 1;
    }

    Node *root = buildQuadtree(rgb, 0, 0, width, height, minBlockSize, threshold,
errorMethod);
    try
    {
        reconstructImage(root, output);
        saveImage(outputPath, output, width, height, extension);
    }
    catch (const exception &e)
    {
        cout << "Terjadi kesalahan: " << e.what() << endl;
        deleteTree(root);
        return 1;
    }

    // waktu eksekusi
    auto end = chrono::high_resolution_clock::now();
    chrono::duration<double> duration = end - start;
    cout << "Waktu eksekusi: " << duration.count() << " detik" << endl;
    chrono::duration<double, milli> duration_ms = end - start;
    cout << "Waktu eksekusi: " << duration_ms.count() << " milidetik" << endl;

    // ukuran gambar sebelum kompresi
    cout << "Hasil Kompresi" << endl;

```

```

        uintmax_t sizeInBytesBefore;
    try
    {
        sizeInBytesBefore = fs::file_size(inputPath);
        double sizeInKBBefore = sizeInBytesBefore / 1024.0;
        double sizeInMBBefore = sizeInKBBefore / 1024.0;
        cout << "Ukuran file sebelum kompresi (dalam KB): " << sizeInKBBefore << " KB" <<
endl;
        cout << "Ukuran file sebelum kompresi (dalam MB): " << sizeInMBBefore << " MB" <<
endl;
    }
    catch (const fs::filesystem_error &e)
    {
        cout << "Gagal membaca ukuran file asli: " << e.what() << endl;
        return 1;
    }

    // ukuran gambar sesudah kompresi
    uintmax_t sizeInBytesAfter = fs::file_size(outputPath);
    double sizeInKBAfter = sizeInBytesAfter / 1024.0;
    double sizeInMBAfter = sizeInKBAfter / 1024.0;
    cout << "Ukuran file hasil kompresi (dalam KB): " << sizeInKBAfter << " KB" << std::endl;
    cout << "Ukuran file hasil kompresi (dalam MB): " << sizeInMBAfter << " MB" << std::endl;

    // presentase kompresi
    uintmax_t compressedSize;
    float compressionPercentage = ((1 - (static_cast<double>(sizeInBytesAfter) /
sizeInBytesBefore)) * 100);
    cout << "Persentase kompresi: " << compressionPercentage << "%" << endl;

    // kedalaman
    int depth = calculateTreeDepth(root);
    cout << "Kedalaman pohon: " << depth << endl;

    // banyak simpul pada pohon
    int nodes = calculateNodeCount(root);
    cout << "Banyak simpul daun: " << nodes << endl;

    // gambar hasil kompresi pada alamat yang sudah ditentukan
    cout << "Gambar hasil disimpan sebagai 'hasil.png' pada " << outputPath << endl;
    deleteTree(root);
}

```

Bab IV

Implementasi Algoritma Divide and Conquer

Pseudocode QuadTree.cpp

Implementasi algoritma Divide and Conquer pada Quadtree.cpp digunakan untuk membangun struktur *quadtree* guna mengompresi dan merekonstruksi gambar berbasis RGB. Pendekatan ini diterapkan dalam fungsi ***buildQuadtree***, dengan memecah gambar menjadi subwilayah yang lebih kecil secara rekursif untuk merepresentasikan data gambar secara efisien. Proses ***divide*** dilakukan dengan membagi wilayah gambar koordinat **(x)**, **(y)**, **(width)**, dan **(height)** menjadi empat subwilayah ***top-left***, ***top-right***, ***bottom-left***, dan ***bottom-right*** dengan ukuran setengah dari wilayah awal. Pembagian ini berlanjut secara rekursif hingga kondisi berhenti terpenuhi, yaitu ketika ukuran wilayah mencapai batas minimum (***minBlockSize***) atau ketika variansi wilayah (Metode Pengukuran Error seperti ***Variance***, ***MAD***, ***MaxPixelDifference***, atau ***Entropy***) berada di bawah ambang batas/*threshold*.

Pada tahap *conquer*, setiap subwilayah diproses untuk menghitung warna rata-rata (R, G, B) menggunakan fungsi ***getAverageColor***, yang menentukan apakah wilayah tersebut menjadi daun (*leaf*) atau perlu dibagi lagi. Jika wilayah menjadi daun, data warna rata-rata disimpan, dan tidak ada pembagian lebih lanjut. Tahap *combine* tidak melibatkan penggabungan eksplisit selama pembangunan quadtree, tetapi direalisasikan saat rekonstruksi gambar melalui fungsi ***reconstructImage***. Fungsi ini secara rekursif mengunjungi setiap simpul *quadtree* dan, jika simpul adalah daun, mengisi wilayah terkait pada gambar keluaran dengan warna rata-rata yang tersimpan. Pendekatan Divide and Conquer memungkinkan representasi gambar yang hemat memori dengan mengelompokkan wilayah homogen ke dalam simpul daun, sementara wilayah dengan variasi warna tinggi dipecah lebih lanjut, sehingga menghasilkan kompresi yang adaptif dan efisien. Fungsi seperti ***calculateTreeDepth*** dan ***calculateNodeCount*** membantu analisis struktur quadtree dengan menghitung kedalaman dan jumlah simpul secara rekursif, yang juga mencerminkan sifat rekursif dari algoritma ini.

Tangkapan layar yang memperlihatkan input dan output (minimal sebanyak 7 buah contoh).

Bab V

Analisis Percobaan Algoritma Divide and Conquer

Case 1

Gambar Input :



Gambar Output :



CLI I/O

```
PS C:\Users\Mahesa\OneDrive\ITB\Coding\College\Academic\IF\Smt-4\Strategi Algoritma\Tucil\Tucil 2\Tucil2_13523134_13523140> ./main
      / \ \ / -- - - - - - - - - - - / \ \ \ / \ / \ \ / \ \ / \ \
     / , < / \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
    / / | / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \
   / / | \ \ / \ / \ / \ . / \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
        \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \
Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\Mahesa\Downloads\images\1.jpg"
Mencoba mengakses: C:\Users\Mahesa\Downloads\images\1.jpg
Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode: 1
Masukkan nilai threshold: 2
Masukkan ukuran blok minimum: 2
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\Mahesa\Downloads\test
Peringatan: Tidak ada ekstensi file. Menggunakan ekstensi dari input: C:\Users\Mahesa\Downloads\test\hasil.jpg
Gambar berhasil disimpan ke: C:\Users\Mahesa\Downloads\test\hasil.jpg
Waktu eksekusi: 5.61214 detik
Waktu eksekusi: 5612.14 milidetik
Hasil Kompresi

Ukuran file sebelum kompresi (dalam KB): 636.267 KB
Ukuran file sebelum kompresi (dalam MB): 0.621354 MB
Ukuran file hasil kompresi (dalam KB): 636.965 KB
Ukuran file hasil kompresi (dalam MB): 0.622036 MB
Persentase kompresi: -0.1097941%
Kedalaman pohon: 11
Banyak simpul daun: 35077
Gambar hasil disimpan pada C:\Users\Mahesa\Downloads\test\hasil.jpg
```

Analisis

- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode Variance, threshold 2, blok minimum 2x2).
- Hasil Kompresi: Dari 636.287 KB jadi 635.965 KB, persentase -0.1097941% (kurang efektif).
- Kompleksitas:
 - waktu: $O(n \log n)$, eksekusi 5.61214 detik.

Ruang: $O(n)$ di kasus terburuk.

Case 2

Input :



Output :

```
+-+ C:\Users\ASUS\Tucil2_13523134_13523140\src> ... bin\program  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/____/  
Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\ASUS\Downloads\IMG_3786.png"  
Mencoba mengakses: C:\Users\ASUS\Downloads\IMG_3786.png  
Pilih metode perhitungan error  
1. Variance  
2. Mean Absolute Deviation (MAD)  
3. Max Pixel Difference  
4. Entropy  
Pilih metode: 1  
Masukkan nilai threshold: 20  
Masukkan ukuran blok minimum: 29  
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\ASUS\Tucil2_13523134_13523140\test  
Peringatan: Anda memasukkan direktori. Output akan disimpan sebagai: C:\Users\ASUS\Tucil2_13523134_13523140\test\hasil.png  
Gambar berhasil disimpan ke: C:\Users\ASUS\Tucil2_13523134_13523140\test\hasil.png  
Waktu eksekusi: 2.94206 detik  
Waktu eksekusi: 2942.06 milidetik  
Hasil Kompresi  
Ukuran file sebelum kompresi (dalam KB): 835.737 KB  
Ukuran file sebelum kompresi (dalam MB): 0.81615 MB  
Ukuran file hasil kompresi (dalam MB): 0.0872879 MB  
Persentase kompresi: 89.3049%  
Kedalaman pohon: 10  
Banyak simpul daun: 47461  
Gambar hasil disimpan sebagai 'hasil.png' pada C:\Users\ASUS\Tucil2_13523134_13523140\test\hasil.png
```

Analisis

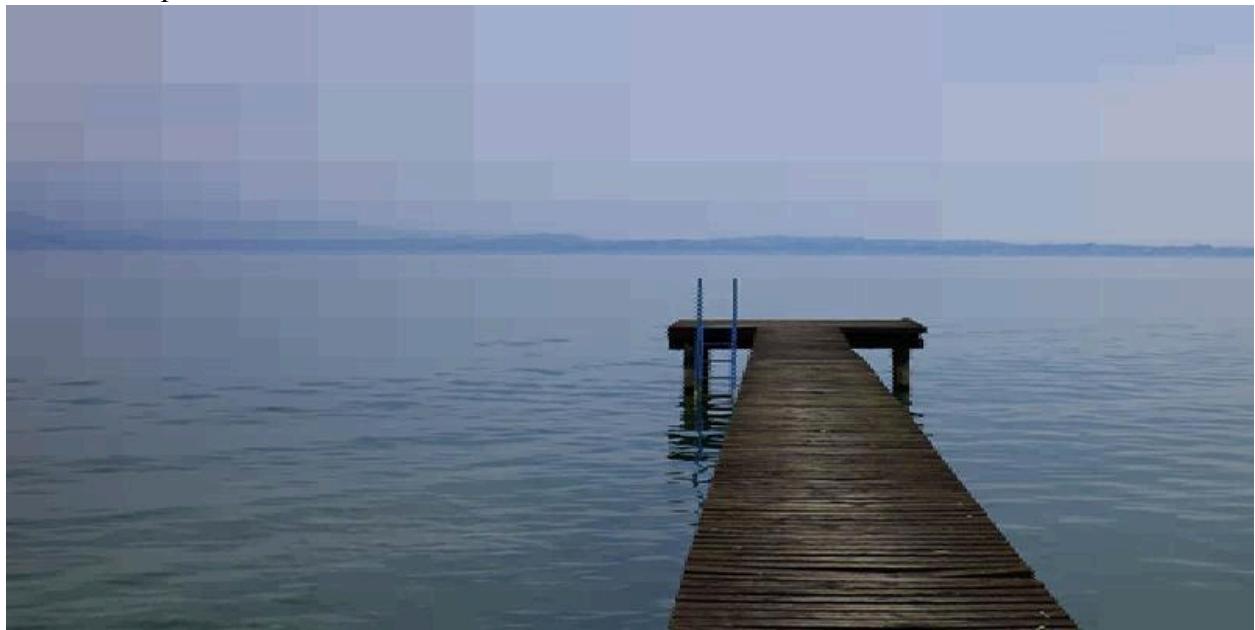
- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode Variance, threshold 20, blok minimum 29x29).
- Hasil Kompresi: Dari 835.737 KB jadi 0.81615 MB, persentase 89.3049% (sangat efektif).
- Kompleksitas:
 - Waktu: $O(n \log n)$, eksekusi 2.94206 detik.
 - Ruang: $O(n)$ di kasus terburuk.

Case 3

Gambar Input :



Gambar Output :



CLI I/O

```

● PS C:\Users\Mahesa\OneDrive\ITB\Coding\College\Academic\IF\Smt-4\Strategi Algoritma\Tucil\Tucil 2\Tucil2_13523134_13523140> ./main
_____
/ ,< / -- \V -- \
/ | / / / / / / / / / / / / / / / / / / / / / / / / / /
/ \ \ / / / / / / . / / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\Mahesa\Downloads\images\2.jpg"
Mencoba mengakses: C:\Users\Mahesa\Downloads\images\2.jpg
Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode: 2
Masukkan nilai threshold: 3
Masukkan ukuran blok minimum: 5
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\Mahesa\Downloads\test
Peringatan: Tidak ada ekstensi file. Menggunakan ekstensi dari input: C:\Users\Mahesa\Downloads\test\hasil.jpg
Gambar berhasil disimpan ke: C:\Users\Mahesa\Downloads\test\hasil.jpg
Waktu eksekusi: 0.982672 detik
Waktu eksekusi: 982.672 millidetik
Hasil Kompresi

Ukuran file sebelum kompresi (dalam KB): 129.992 KB
Ukuran file sebelum kompresi (dalam MB): 0.126945 MB
Ukuran file hasil kompresi (dalam KB): 125.498 KB
Ukuran file hasil kompresi (dalam MB): 0.122557 MB
Persentase kompresi: 3.45724%
Kedalaman pohon: 10
Banyak simpul daun: 58157
Gambar hasil disimpan pada C:\Users\Mahesa\Downloads\test\hasil.jpg

```

Analisis

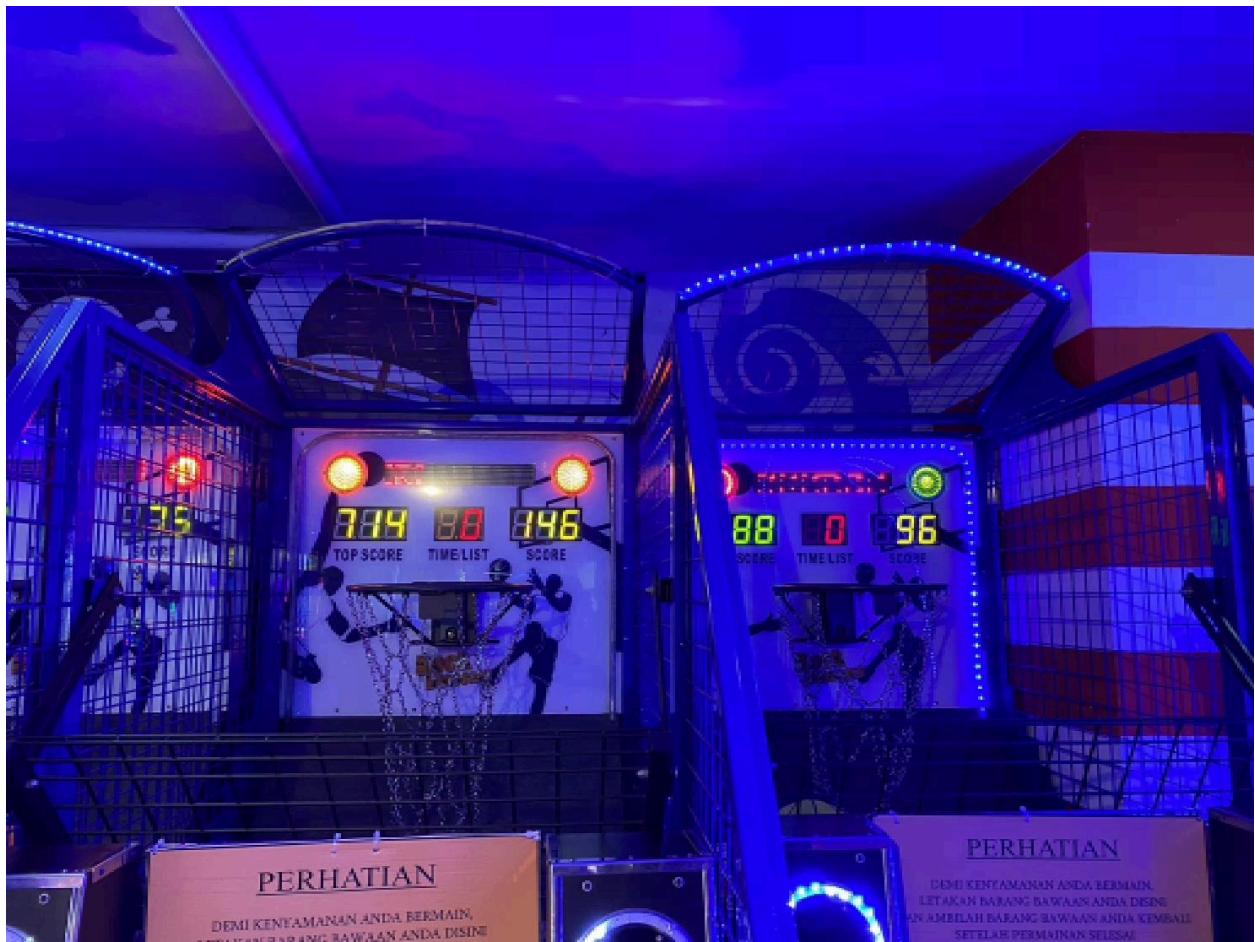
- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode MAD, threshold 3, blok minimum 5x5).
- Hasil Kompresi: Dari 129.992 KB jadi 125.498 KB, persentase 3.45724% (cukup efektif).
- Kompleksitas:
 - Waktu: $O(n \log n)$, eksekusi 0.982672 detik.
 - Ruang: $O(n)$ di kasus terburuk.

Case 4

Gambar Input :



Gambar Output :



CLI I/O

```

PS C:\Users\Mahesa\OneDrive\ITB\Coding\College\Academic\IF\Smt-4\Strategi Algoritma\Tucil\Tucil 2\Tucil2_13523134_13523140> ./main
_____
/ ,< / _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \ V _ \ _ \
/ | / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / | \ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
/ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / /
Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\Mahesa\Downloads\S__16564283.jpg"
Mencoba mengakses: C:\Users\Mahesa\Downloads\S__16564283.jpg
Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode: 3
Masukkan nilai threshold: 5
Masukkan ukuran blok minimum: 10
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\Mahesa\Downloads\test
Peringatan: Tidak ada ekstensi file. Menggunakan ekstensi dari input: C:\Users\Mahesa\Downloads\test\hasil.jpg
Gambar berhasil disimpan ke: C:\Users\Mahesa\Downloads\test\hasil.jpg
Waktu eksekusi: 4.56285 detik
Waktu eksekusi: 4562.85 milidetik
Hasil Kompresi

Ukuran file sebelum kompresi (dalam KB): 1668.6 KB
Ukuran file sebelum kompresi (dalam MB): 1.62949 MB
Ukuran file hasil kompresi (dalam KB): 1342.48 KB
Ukuran file hasil kompresi (dalam MB): 1.31102 MB
Persentase kompresi: 19.5442%
Kedalaman pohon: 10
Banyak simpul daun: 217501
Gambar hasil disimpan pada C:\Users\Mahesa\Downloads\test\hasil.jpg

```

Analisis

- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode Max Pixel Difference, threshold 5, blok minimum 10x10).
- Hasil Kompresi: Dari 1668.6 KB jadi 1342.48 KB, persentase 19.5442% (efektif).
- Kompleksitas:
 - Waktu: $O(n \log n)$, eksekusi 4.56285 detik.
 - Ruang: $O(n)$ di kasus terburuk.

Case 5

Gambar Input :



Gambar Output :



CLI I/O

```
PS C:\Users\Mahesa\OneDrive\ITB\Coding\College\Academic\IF\Smt-4\Strategi Algoritma\Tucil\Tucil 2\Tucil2_13523134_13523140> ./main
      / \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
     / < \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ \ 
    / / \ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 
   / / \ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 
  / / \ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 
 / / \ / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / / 
Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\Mahesa\Downloads\images\mario.jpeg"
Mencoba mengakses: C:\Users\Mahesa\Downloads\images\mario.jpeg
Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode: 4
Masukkan nilai threshold: 2
Masukkan ukuran blok minimum: 2
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\Mahesa\Downloads\test
Peringatan: Tidak ada ekstensi file. Menggunakan ekstensi dari input: C:\Users\Mahesa\Downloads\test\hasil.jpeg
Gambar berhasil disimpan ke: C:\Users\Mahesa\Downloads\test\hasil.jpeg
Waktu eksekusi: 0.114562 detik
Waktu eksekusi: 114.562 milidetik
Hasil Kompresi

Ukuran file sebelum kompresi (dalam KB): 12.2002 KB
Ukuran file sebelum kompresi (dalam MB): 0.0119143 MB
Ukuran file hasil kompresi (dalam KB): 6.0006 KB
Ukuran file hasil kompresi (dalam MB): 0.058692 MB
Persentase kompresi: -392.62%
Kedalaman pohon: 8
Banyak simpul daun: 10977
Gambar hasil disimpan pada C:\Users\Mahesa\Downloads\test\hasil.jpeg
```

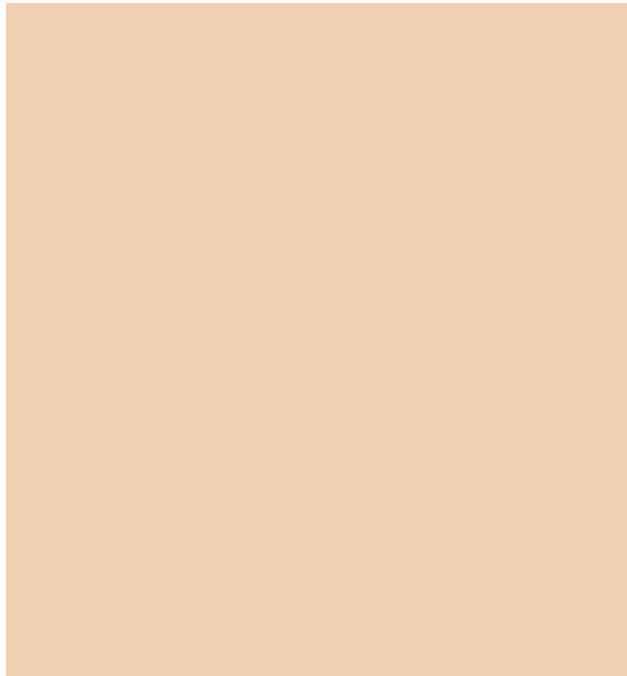
Analisis

- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode Entropy, threshold 2, blok minimum 2x2).
- Hasil Kompresi: Dari 12.2802 KB jadi 6.85869 KB, persentase 39.2% (sangat efektif).
- Kompleksitas:
 - Waktu: $O(n \log n)$, eksekusi 0.114562 detik.
 - Ruang: $O(n)$ di kasus terburuk.

Case 7



Gambar Input :



Gambar Output :

CLI I/O

```

    / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / 
    / \ | \ \ \ / \ / \ / . \ / \ / \ \ \ / \ / \ / \ \ / \ / \ / \ \ / \ / \ / \ 
    / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ / \ 

Input alamat absolut file gambar yang ingin dikompresi (inputPath): "C:\Users\ASUS\Downloads\Burpy.jpeg"
Mencoba mengakses: C:\Users\ASUS\Downloads\Burpy.jpeg
Pilih metode perhitungan error
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode: 4
Masukkan nilai threshold: 40
Masukkan ukuran blok minimum: 34
Input alamat absolut untuk file gambar yang sudah dikompresi (outputPath): C:\Users\ASUS\Downloads\hasil
Peringatan: Anda memasukkan direktori. Output akan disimpan sebagai: C:\Users\ASUS\Downloads\hasil\hasil.png
Gambar berhasil disimpan ke: C:\Users\ASUS\Downloads\hasil\hasil.png
Waktu eksekusi: 0.0406164 detik
Waktu eksekusi: 40.6164 milidetik
Hasil Kompresi
Ukuran file sebelum kompresi (dalam KB): 13.2451 KB
Ukuran file sebelum kompresi (dalam MB): 0.0129347 MB
Ukuran file hasil kompresi (dalam KB): 3.46387 KB
Ukuran file hasil kompresi (dalam MB): 0.00338268 MB
Persentase kompresi: 73.848%
Kedalaman pohon: 1
Banyak simpul daun: 1
Gambar hasil disimpan sebagai 'hasil.png' pada C:\Users\ASUS\Downloads\hasil\hasil.png
PS C:\Users\ASUS\Tucil2_13523134_13523140\src> █

```

Analisis

- Divide and Conquer Quadtree: Gambar dibagi rekursif jadi blok homogen (metode Entropy, threshold 40, blok minimum 34x34).
- Hasil Kompresi: Dari 13.2451 KB jadi 3.46387 KB, persentase 73.848% (sangat efektif).
- Kompleksitas:
 - Waktu: $O(n \log n)$, eksekusi 0.046164 detik.
 - Ruang: $O(n)$ di kasus terburuk.

\

Bab VI

Kesimpulan dan Saran

Hasil analisis percobaan algoritma divide and conquer dalam kompresi gambar dengan metode Quadtree. Kompleksitas Waktu $O(n \log n)$ untuk eksekusi.

Bab VII

Lampiran

Repository Github : https://github.com/iannn23/Tucil2_13523134_13523140.git

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