

IF2211 – Strategi Algoritma

Tugas Kecil 2

Kompresi Gambar Dengan Metode Quadtree



Disusun Oleh:

SALMAN HANIF

13523056

Program Studi Teknik Informatika

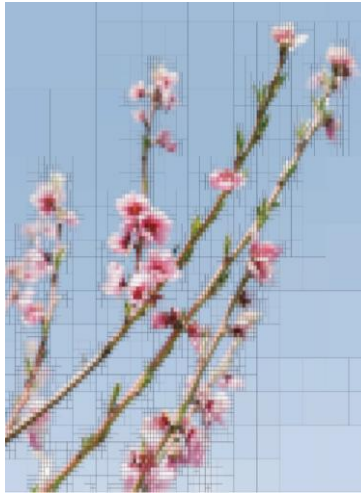
Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

Bandung 2025

BAB I

DESKRIPSI TUGAS



Gambar 1 Quadtree dalam Kompresi Gambar

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

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

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

BAB II

ALGORITMA DIVIDE AND CONQUER

2.1. Langkah-Langkah Ide Algoritma

Setelah dilakukan import image dan dibuat ke bentuk QuadTree, langkah selanjutnya sebagai berikut (mainProcess):

1. Menghitung Variansi/Ukuran Ketidakhomogenan

Ukuran ketidakhomogenan dihitung dari blok citra saat ini berdasarkan parameter:

1: Menggunakan variansi

2: Menggunakan Mean Absolute Deviation (MAD)

3: Menggunakan Mean Pixel Difference (MPD)

4: Menggunakan entropi

2. Memeriksa *Base Case*

- Jika nilai variance lebih kecil dari threshold yang telah ditentukan.
- Jika luas blok atau seperempat luas block kurang dari minimum block yang ditentukan.

Jika salah satu kondisi terpenuhi, blok dinyatakan sebagai leaf dan nilai pixel rata-rata dari blok dihitung.

3. Pembagian Blok (Divide Step)

Jika blok tidak homogen dan ukuran blok cukup besar:

- Tentukan titik tengah blok berdasarkan lebar (midWidth) dan tinggi (midHeight).

Blok dipotong 4 sama besar (kiri atas, kanan atas, kiri bawah, kanan bawah)

- Sub-blok 1: Posisi poinnya sama dengan titik awal (pojok kiri atas).
- Sub-blok 2: Pindah ke kanan sebanyak midWidth.
- Sub-blok 3: Pindah ke bawah sebanyak midHeight.

Sub-blok 4: Pindah ke kanan dan ke bawah sebesar midWidth dan midHeight

- Sesuaikan ukuran sub-blok untuk mengantisipasi kondisi dimana lebar atau tinggi blok adalah bilangan ganjil (jika lebar ganjil, tambah 1 pixel untuk blok yang berada di kanan, jika tinggi ganjil, tambah 1 pixel untuk blok yang di bawah)

4. Membuat Sub-Blok (Conquer Step)

Untuk masing-masing sub-blok, buat objek baru dari kelas QuadTree dengan posisi dan data blok yang sesuai.

5. Rekursi pada Setiap Child

Untuk setiap empat child node yang baru dibuat, panggil kembali fungsi ini secara rekursif.

Setelah rekursi berhenti (semua sub-block telah didapatkan leaf), maka proses selanjutnya adalah image disusun dengan informasi pixel dari masing-masing leaf sesuai posisinya masing-masing dan diekstrak menjadi satu image utuh.

2.2.Pseudocode Algoritma Divide and Conquer

```
FUNCTION mainProcess(emd, threshold, minblock, block, position)
{ 1. Menghitung ukuran ketidakhomogenan (variance) berdasarkan parameter EMD }
  IF emd == 1 THEN
    variance ← calculateVariance(block)
  ELSE IF emd == 2 THEN
    variance ← calculateMAD(block)
  ELSE IF emd == 3 THEN
    variance ← calculateMPD(block)
  ELSE
    variance ← calculateEntropy(block)
  END IF

{ 2. Memeriksa Base Case }
  IF (variance < threshold) OR (block.area < minblock) OR ((block.area / 4) < minblock)
  THEN
    averageColor ← calculateAverageColor(block)
    // Tandai blok sebagai leaf dengan menyimpan rata-rata warnanya
    block.setAverageColor(averageColor)
    RETURN
  END IF

{ 3. Pembagian Blok (Divide Step) }
  midWidth ← FLOOR(block.width / 2)
  midHeight ← FLOOR(block.height / 2)

{Penyesuaian ukuran untuk kasus bilangan ganjil}
  IF (block.width MOD 2 ≠ 0) THEN
    w0 ← midWidth
    w1 ← midWidth + 1
```

```

ELSE
    w0 ← midWidth
    w1 ← midWidth
END IF

IF (block.height MOD 2 ≠ 0) THEN
    h0 ← midHeight
    h1 ← midHeight + 1
ELSE
    h0 ← midHeight
    h1 ← midHeight
END IF

// Menentukan posisi untuk masing-masing sub-blok
pos1 ← position // Pojok kiri atas
pos2 ← (position.x + midWidth, position.y) // Kanan atas
pos3 ← (position.x, position.y + midHeight) // Kiri bawah
pos4 ← (position.x + midWidth, position.y + midHeight) //
Kanan bawah

// 4. Membuat Sub-Blok (Conquer Step)
subBlock0 ← block.getSubBlock(0, 0, w0, h0) //
Sub-blok kiri atas
subBlock1 ← block.getSubBlock(midWidth, 0, w1, h0)
// Sub-blok kanan atas
subBlock2 ← block.getSubBlock(0, midHeight, w0, h1)
// Sub-blok kiri bawah
subBlock3 ← block.getSubBlock(midWidth, midHeight, w1, h1)
// Sub-blok kanan bawah

// Membuat empat child QuadTree berdasarkan sub-blok
child0 ← new QuadTree(pos1, subBlock0)
child1 ← new QuadTree(pos2, subBlock1)
child2 ← new QuadTree(pos3, subBlock2)
child3 ← new QuadTree(pos4, subBlock3)
// 5. Rekursi pada Setiap Child
FOR i FROM 0 TO 3 DO
    child[i].mainProcess(emd, threshold, minblock)
END FOR
END FUNCTION

```

BAB III

SOURCE CODE

Untuk source lengkap yang memuat fungsi implementasi cpp dapat dilihat pada link github. Di sini hanya ditampilkan main dan header dari tiap class.

3.1 Point.hpp

```
// Class Point merepresentasikan sebuah titik/koordinat (x, y) di dalam citra atau bidang 2D
class Point
{
private:
    // Menyimpan posisi horizontal (x) dan vertikal (y)
    int x;
    int y;

public:
    // Konstruktor default
    // Menginisialisasi titik ke koordinat (0, 0)
    Point();

    // Konstruktor dengan parameter
    // Menginisialisasi titik ke koordinat (x, y) sesuai input
    Point(int x, int y);

    // Destructor
    ~Point();

    // Mengembalikan nilai posisi x dari titik
    int getX();

    // Mengembalikan nilai posisi y dari titik
    int getY();

    // Menampilkan posisi titik dalam format (x, y)
    // Biasanya digunakan untuk debugging
    void displayPosition();
}
```

3.2 Pixel.hpp

```
// Class Pixel merepresentasikan sebuah pixel (titik) warna dalam citra
class Pixel
{
private:
    // Menyimpan nilai warna merah (red), hijau (green), dan biru (blue)
    unsigned char red, green, blue;

public:
    // Konstruktor default
    // Menginisialisasi nilai RGB pixel ke 0 (hitam)
    Pixel();

    // Konstruktor dengan parameter
    // Menginisialisasi nilai RGB pixel sesuai input
    // r = nilai Red, g = nilai Green, b = nilai Blue
    Pixel(int r, int g, int b);

    // Destructor
    // Tidak ada resource khusus yang dibebaskan
    ~Pixel();

    // Mengembalikan nilai warna pixel berdasarkan colourcode
    // colourcode: 0 = Red, 1 = Green, 2 = Blue
    int getColour(int colourcode);

    // Mengubah nilai warna pixel berdasarkan colourcode
    // value: nilai baru untuk warna tersebut
    void setColour(int colourcode, double value);

    // Menampilkan nilai warna RGB pixel
    void displayPixel();
}
```

3.3 Block.hpp

```
// Class Block merepresentasikan sebuah blok atau area persegi panjang dalam gambar
// Digunakan sebagai elemen dasar dalam proses Quadtree compression

class Block
{
private:
    // Lebar blok (jumlah kolom)
    int width;

    // Tinggi blok (jumlah baris)
    int height;

    // Matriks 2D yang menyimpan nilai warna (Pixel) untuk tiap posisi dalam blok
    Pixel** intensity;

public:
    // Konstruktor default
    Block();

    // Konstruktor dengan parameter
    // Membuat blok dengan ukuran (width x height)
    Block(int w, int h);

    // Copy constructor
    // Digunakan untuk menyalin isi blok dari blok lain
    Block(const Block& other);

    // Operator assignment
    // Meng-assign blok lain ke blok ini
    Block& operator=(const Block& other);

    // Destructor
    // Menghapus alokasi memori dari matriks intensity
    ~Block();

    // Mengembalikan lebar blok
    int getWidth();
```



```
// Mengembalikan tinggi blok
int getHeight();

// Mengembalikan luas area blok (width * height)
int getArea();

// Menampilkan isi blok (biasanya untuk debugging)
void displayBlock();

// Mengembalikan matriks 2D intensity dari blok
Pixel** getBlockIntensity() const;

// Mengambil sub-block dari posisi tertentu (startPoint) dengan ukuran (w x h)
// Digunakan saat proses pembagian (divide) Quadtree
Block* getSubBlock(Point startPoint, int w, int h);

// Mengambil pixel di posisi tertentu
Pixel getIntensity(Point pos) const;

// Mengatur nilai pixel di posisi tertentu
void setIntensity(Point pos, const Pixel& pixel);

// Mengisi seluruh blok dengan warna tertentu
void fillWithColor(const Pixel &pix);

// Melakukan normalisasi nilai warna di blok (jika diperlukan untuk preprocessing)
void normalise();

// Menghitung rata-rata nilai warna untuk channel tertentu (0:Red, 1:Green, 2:Blue)
double getAverage(int colourCode);

// Menghitung rata-rata warna dari seluruh blok (hasilnya berupa Pixel RGB)
Pixel calculateAverageColor();

// Menghitung variance dari seluruh blok (ukuran ketidakhomogenan)
double calculateVariance();

// Fungsi helper untuk calculateVariance per channel warna
double calculateVarianceHelper(int colourCode);

// Menghitung Mean Absolute Deviation (MAD) dari blok
double calculateMAD();
```

```

// Fungsi helper untuk calculateMAD per channel warna
double calculateMADHelper(int colourCode);

// Menghitung Mean Pixel Difference (MPD) dari blok
double calculateMPD();

// Fungsi helper untuk calculateMPD per channel warna
std::pair<int, int> calculateMPDHelper(int colourCode);

// Menghitung Entropy dari blok (ukuran kompleksitas pola warna)
double calculateEntropy();

// Fungsi helper untuk calculateEntropy per channel warna
double calculateEntropyHelper(int colourCode);
};

#endif

```

3.4 Quadtree.hpp

```

#ifndef QUADTREE_H
#define QUADTREE_H

// Class QuadTree digunakan untuk merepresentasikan struktur data Quadtree
// yang berfungsi untuk kompresi gambar berbasis pembagian area menjadi blok-blok lebih kecil.

class QuadTree
{
private:
    // Koordinat posisi pojok kiri atas dari blok ini dalam gambar
    Point position;

    // Blok gambar yang direpresentasikan oleh node Quadtree ini
    Block block;

    // Array pointer ke anak-anak (sub-blok) Quadtree (maksimal 4 anak)
    QuadTree** child;

    // Maksimal anak dari setiap node Quadtree (tetap 4 sesuai konsep Quadtree)
    int maxChild = 4;

    // Rata-rata warna pixel dari blok (dipakai kalau blok jadi leaf / tidak dibagi lagi)
    Pixel average;

```

```

// Konstruktor
// Membuat node Quadtree dengan posisi dan blok tertentu
QuadTree(Point pos, Block sz);

// Membuat Quadtree dari sebuah file gambar
static QuadTree* buildFromImage (const char* filename);

// Proses utama kompresi Quadtree
// emd -> metode perhitungan homogenitas (1: Variance, 2: MAD, 3: M
// threshold -> batas homogenitas
// minSize -> ukuran minimum blok untuk bisa dibagi
void mainProcess(int emd, double threshold, int minSize);

// Destructor
// Menghapus semua alokasi memori dari anak-anak Quadtree
~QuadTree();

// Menyimpan hasil Quadtree ke file
void saveToImage(std::ofstream &out);

// Membaca dan membangun kembali Quadtree dari file
static QuadTree* loadFromImage(std::ifstream &in);

// Mengisi canvas kosong dengan warna dari Quadtree (untuk rebuild
void fillCanvas(Pixel** canvas);

// Membuat Quadtree dummy untuk testing (tanpa file image)
QuadTree* buildDummyTree(int width, int height);

// Menampilkan struktur Quadtree (biasanya untuk debugging)

```

```

// Menampilkan struktur Quadtree (biasanya untuk debugging)
void printQuadTree();

// Mengembalikan kedalaman (depth) dari Quadtree
int getDepth();

// Menghitung jumlah node total dari Quadtree
int countNodes();

// Mengembalikan lebar dari blok Quadtree saat ini
int getWidth();

// Mengembalikan tinggi dari blok Quadtree saat ini
int getHeight();

```

3.5 Main.cpp

```
#include <iostream>
#include <chrono>
#include <vector>
#include <fstream>
#include <filesystem>
#include "quadtree.hpp"
#include <string>

using namespace std;
namespace fs = filesystem;

#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

std::vector<unsigned char> load_file(const std::string& filename) {
    std::ifstream file(filename, std::ios::binary);
    return std::vector<unsigned char>((std::istreambuf_iterator<char>(file)), std::istreambuf_iterator<char>());
}

double inputThreshold(double minVal, double maxVal, string validRange) {
    double threshold;
    while (true) {
        cout << endl << "Nilai Threshold memungkinkan : " << minVal << " - " << maxVal << endl;
        cout << "Nilai Threshold ideal : " << validRange << endl;
        cout << "Tentukan Nilai Threshold: ";
        cin >> threshold;

        if (cin.fail() || threshold < minVal || threshold > maxVal) {
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
            cout << "Input threshold tidak valid, masukkan antara " << minVal << " dan " << maxVal << "." << endl;
        } else {
            break;
        }
    }
    return threshold;
}

string inputOutputPath() {
    string path;
    cout << "Tuliskan filepath untuk hasil (output image): ";
    cin >> path;
}
```

```

// Cek apakah file sudah ada
if (fs::exists(path)) {
    char pilihan;
    cout << "File sudah ada! Apakah ingin menimpa file tersebut? (y/n): ";
    cin >> pilihan;

    while (pilihan != 'y' && pilihan != 'Y' && pilihan != 'n' && pilihan != 'N') {
        cout << "Input tidak valid. Masukkan 'y' atau 'n': ";
        cin >> pilihan;
    }

    if (pilihan == 'n' || pilihan == 'N') {
        cout << "Silakan masukkan filepath yang berbeda: ";
        return inputOutputPath(); // rekursif sampai dapet path baru
    }
}

return path;
}

void printSummary(const string& in, const string& out, int emd, double threshold, int minSize) {
    cout << "===== RINGKASAN INPUT =====" << endl;
    cout << "File input image   : " << in << endl;
    cout << "Metode Error          : " << emd << endl;
    cout << "Threshold             : " << threshold << endl;
    cout << "Minimal Block Size : " << minSize << endl;
    cout << "===== " << endl << endl;
}

int main(int argc, char const *argv[])
{
    // Input dari user:
    string imagepathIn;
    int emd;
    double threshold;
    int minSize;
    string imagepathOut;

    cout << endl << "~Welcome to Image Compressor by QuadTree~" << endl << endl;
    cout << "Tuliskan filepath dari Image (input): ";
    cin >> imagepathIn;
    // Validasi Error Measurement Method (EMD)

```

```

while (true) {
    cout << "\nError Measurement Method" << endl;
    cout << "1. Variance" << endl;
    cout << "2. Mean Absolute Deviation" << endl;
    cout << "3. Max Pixel Difference" << endl;
    cout << "4. Entropy" << endl;
    cout << "Choose one (number): ";
    cin >> emd;

    if (cin.fail() || emd < 1 || emd > 4) {
        cin.clear(); // Reset stream error flag
        cin.ignore(numeric_limits<streamsize>::max(), '\n'); // Singkirkan input yang salah
        cout << "Input tidak valid. Silakan masukkan angka 1, 2, 3 atau 4." << endl;
    } else {
        break;
    }
}

// Validasi input untuk threshold berdasarkan EMD yang dipilih
bool validThreshold = false;
if (emd == 1) {
    threshold = inputThreshold(0, 16256, "50 - 500");
} else if (emd == 2) {
    threshold = inputThreshold(0, 127.5, "5 - 30");
} else if (emd == 3) {
    threshold = inputThreshold(0, 255, "10 - 50");
} else {
    threshold = inputThreshold(0, 8, "0.5 - 3");
}

// Validasi input untuk Area Minimum Block (harus > 0)
while (true) {
    cout << "\nTentukan Area Minimum Block: ";
    cin >> minSize;
    if (cin.fail() || minSize <= 0) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input area minimum block tidak valid. Silakan masukkan angka positif." << endl;
    } else {
        break;
    }
}

imagepathOut = inputOutputPath();

```

```

// Clear layar
system("cls");
printSummary(imagepathIn, imagepathOut, emd, threshold, minSize);

cout << "Processing..." << endl << endl;
// 1. Membangun quadtree dari gambar menggunakan fungsi impor buildFromImage
QuadTree* tree = QuadTree::buildFromImage(imagepathIn.c_str());
if(tree == nullptr){
    cerr << "Gagal membangun QuadTree dari gambar!" << endl;
    return 1;
}

// 2. Melakukan proses kompresi dengan subdivisi (sesuai error measurement, threshold, dan minimum block)
auto start = chrono::high_resolution_clock::now();
tree->mainProcess(emd, threshold, minSize);

// 3. Serialisasi struktur quadtree ke file biner
ofstream outFile("compressed.quadtree", ios::binary);
if (!outFile) {
    cerr << "Gagal membuka file untuk penulisan quadtree!" << endl;
    return 1;
}
tree->saveToImage(outFile);
outFile.close();

// 4. Deserialisasi quadtree dari file untuk merekonstruksi gambar
ifstream inFile("compressed.quadtree", ios::binary);
if (!inFile) {
    cerr << "Gagal membuka file untuk pembacaan quadtree!" << endl;
    return 1;
}
QuadTree* loadedTree = QuadTree::loadFromImage(inFile);
inFile.close();

// 5. Rekonstruksi canvas dari quadtree yang dideserialisasi
int width = loadedTree->getWidth();
int height = loadedTree->getHeight();

```

```

Pixel** canvas = new Pixel*[height];
for (int i = 0; i < height; i++) {
    canvas[i] = new Pixel[width];
}
// Inisialisasi canvas dengan warna default (hitam)
// for (int y = 0; y < height; y++) {
//     for (int x = 0; x < width; x++) {
//         canvas[y][x] = Pixel(0,0,0);
//     }
// }
loadedTree->fillCanvas(canvas);

// 6. Export canvas hasil rekonstruksi ke file gambar
unsigned char* imgData = new unsigned char[width * height * 3];
for (int y = 0; y < height; y++) {
    for (int x = 0; x < width; x++) {
        int idx = (y * width + x) * 3;
        imgData[idx] = canvas[y][x].getColour(0);
        imgData[idx + 1] = canvas[y][x].getColour(1);
        imgData[idx + 2] = canvas[y][x].getColour(2);
    }
}
stbi_write_png(imagepathOut.c_str(), width, height, 3, imgData, width * 3);

// 7. Menghitung waktu eksekusi dan perbandingan ukuran file
auto end = chrono::high_resolution_clock::now();
auto exec_time = chrono::duration<double>(end-start).count();

auto bufferIn = load_file(imagepathIn);
auto bufferOut = load_file(imagepathOut);
int widthIn, heightIn, channelsIn;
int widthOut, heightOut, channelsOut;
stbi_info_from_memory(bufferIn.data(), bufferIn.size(), &widthIn, &heightIn, &channelsIn);
stbi_info_from_memory(bufferOut.data(), bufferOut.size(), &widthOut, &heightOut, &channelsOut);
uintmax_t sizeIn = filesystem::file_size(imagepathIn);
uintmax_t sizeOut = filesystem::file_size(imagepathOut);
double cp = (1 - (double)sizeOut/(double)sizeIn) * 100;

cout << "\nCompressing Complete!" << endl;
cout << "DETAILS:" << endl;
cout << "Execution Time      : " << exec_time*1000 << " ms" << endl;
cout << "Size before (input)   : " << sizeIn/1000 << " kb, " << widthIn << "x" << heightIn << " pixels" << endl;
cout << "Size after (output)   : " << sizeOut/1000 << " kb, " << widthOut << "x" << heightOut << " pixels" << endl;
cout << "Compression Percent   : " << fixed << setprecision(3) << cp << " %" << endl;
cout << "Size before (input)   : " << sizeIn/1000 << " kb, " << widthIn << "x" << heightIn << " pixels" << endl;
cout << "Size after (output)   : " << sizeOut/1000 << " kb, " << widthOut << "x" << heightOut << " pixels" << endl;
cout << "Compression Percent   : " << fixed << setprecision(3) << cp << " %" << endl;
cout << "Tree's Depth          : " << tree->getDepth() << endl;
cout << "Nodes Count           : " << tree->countNodes() << endl;
cout << "Result Path           : " << imagepathOut << endl;

// 8. Cleanup memori
for (int i = 0; i < height; i++) {
    delete[] canvas[i];
}
delete[] canvas;
delete[] imgData;
delete tree;
delete loadedTree;

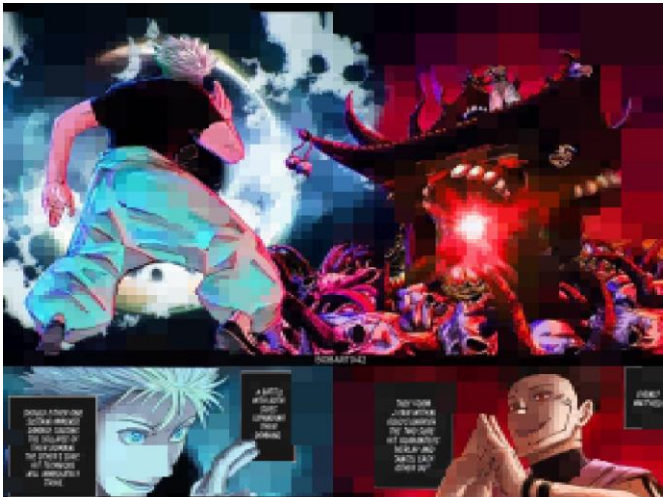
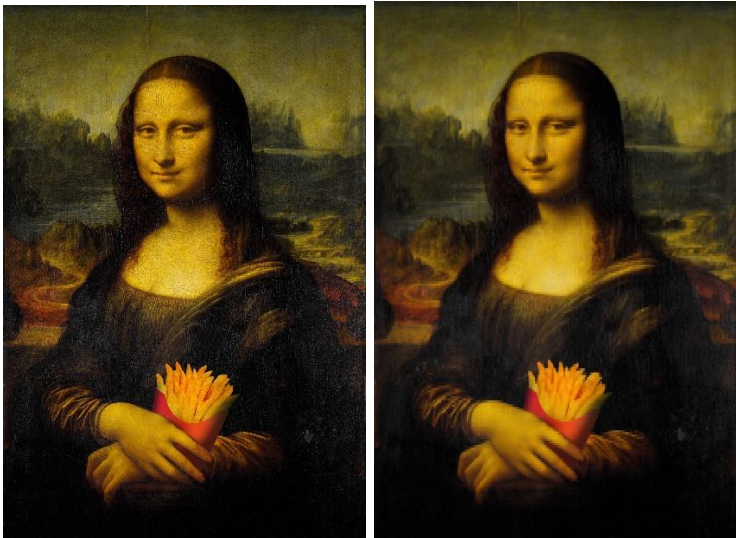

return 0;
}

```


BAB 4

TEST CASE

Test Case	Hasil	
<pre> ===== RINGKASAN INPUT ===== File input image : C:\Users\saman\Documents\Code\Stima\Tucil\ Metode Error : 1 Threshold : 50 Minimal Block Size : 16 ===== Processing... Compressing Complete! DETAILS: Execution Time : 972.538 ms Size before (input) : 126 kb, 728x1294 pixels Size after (output) : 76 kb, 728x1294 pixels Compression Percent : 39.241 % Tree's Depth : 9 Nodes Count : 13689 Result Path : C:\Users\saman\Documents\Code\Stima\Tucil\ PS C:\Users\saman\Documents\Code\C++\Stima\Tucil\ </pre> <p>Variance</p>	Sebelum	Sesudah
<pre> ===== RINGKASAN INPUT ===== File input image : C:\Users\saman\Documents\Code\Stima\Tucil\ Metode Error : 2 Threshold : 15 Minimal Block Size : 12 ===== Processing... Compressing Complete! DETAILS: Execution Time : 832.662 ms Size before (input) : 183 kb, 1170x898 pixels Size after (output) : 154 kb, 1170x898 pixels Compression Percent : 15.812 % Tree's Depth : 9 Nodes Count : 31277 Result Path : C:\Users\saman\Documents\Code\Stima\Tucil\ es.jpg PS C:\Users\saman\Documents\Code\C++\Stima\Tucil\ </pre> <p>MAD</p>	Sebelum	

	<div>Sesudah</div> 
<pre> ===== RINGKASAN INPUT ===== File input image : C:\Users\saman\Documents\C Metode Error : 3 Threshold : 20 Minimal Block Size : 14 ===== Processing... Compressing Complete! DETAILS: Execution Time : 2756.24 ms Size before (input) : 544 kb, 1288x1920 pixels Size after (output) : 297 kb, 1288x1920 pixels Compression Percent : 45.293 % Tree's Depth : 9 Nodes Count : 82269 Result Path : C:\Users\saman\Documents\ es.jpg </pre> <div>MPD</div>	<div>Sebelum Sesudah</div> 
<pre> ===== RINGKASAN INPUT ===== File input image : C:\Users\saman\Documents\Cod Metode Error : 1 Threshold : 100 Minimal Block Size : 13 ===== Processing... Compressing Complete! DETAILS: Execution Time : 1542.24 ms Size before (input) : 1213 kb, 1100x1615 pixels Size after (output) : 126 kb, 1100x1615 pixels Compression Percent : 89.561 % Tree's Depth : 9 Nodes Count : 19169 Result Path : C:\Users\saman\Documents\Cod es.png </pre> <div>Variance</div>	<div>Sebelum : Sesudah :</div> 

```
===== RINGKASAN INPUT =====
File input image : C:\Users\saman\Documents\
Metode Error : 4
Threshold : 0.5
Minimal Block Size : 3
=====

Processing...

Compressing Complete!
DETAILS:
Execution Time : 1872.14 ms
Size before (input) : 867 kb, 691x920 pixels
Size after (output) : 449 kb, 691x920 pixels
Compression Percent : 48.229 %
Tree's Depth : 10
Nodes Count : 186225
Result Path : C:\Users\saman\Document
es.png
```

Entropy

Sebelum



Sesudah



```
===== RINGKASAN INPUT =====
File input image : C:\Users\saman\Documents\Code
Metode Error : 4
Threshold : 3
Minimal Block Size : 16
=====

Processing...

Compressing Complete!
DETAILS:
Execution Time : 9593.08 ms
Size before (input) : 944 kb, 5105x2871 pixels
Size after (output) : 556 kb, 5105x2871 pixels
Compression Percent : 41.043 %
Tree's Depth : 10
Nodes Count : 44981
Result Path : C:\Users\saman\Documents\Co
es.jpg
```

Entropy

Sebelum



Sesudah




```
===== RINGKASAN INPUT =====
File input image   : C:\Users\saman\Documents\Cod
Metode Error       : 3
Threshold          : 10
Minimal Block Size : 10
=====

Processing...

Compressing Complete!
DETAILS:
Execution Time     : 23363.3 ms
Size before (input) : 4890 kb, 5616x3744 pixels
Size after (output) : 2893 kb, 5616x3744 pixels
Compression Percent : 40.844 %
Tree's Depth       : 11
Nodes Count        : 851113
Result Path        : C:\Users\saman\Documents\Cod
es.jpg
```

MPD

Sebelum



Sesudah



BAB 5

ANALISIS ALGORITMA

Kompleksitas Algoritma

Kompleksitas waktu dari algoritma ini sangat bergantung pada tingkat kehomogenan gambar dan nilai threshold yang digunakan. Pada kebanyakan kasus yaitu gambar tidak homogen (sehingga hampir seluruh area gambar perlu dibagi hingga ukuran terkecil), maka kompleksitas waktu dapat dituliskan sebagai:

$$T(n) = 4T\left(\frac{n}{2}\right) + O(n^2)$$

Kasus ini termasuk dalam:

$$O(n^2 \log n)$$

Faktor-Faktor yang Mempengaruhi Performa:

- Nilai *threshold* : Semakin kecil threshold, semakin dalam Quadtree terbentuk → waktu dan size lebih besar.
- Parameter *minBlock* : Semakin kecil minBlock, semakin dalam rekursi berjalan.
- Jenis gambar : Gambar dengan warna seragam (homogen) akan lebih cepat dikompresi dibanding gambar dengan banyak detail.

Analisis Hasil dari Tiap Error Measurement Method :

- Variance
Metode Variance mampu menggambarkan detail dan akurasi warna dengan baik. Tetapi, metode ini kesulitan dalam menangkap gradasi warna sehingga area dengan perubahan warna seringkali dianggap sebagai satu keseragaman dan dijadikan 1 block.
- Mean Absolute Deviation (MAD)
Secara umum, hasil yang didapat dari metode ini memiliki karakteristik yang sama dengan variance
- Mean Pixel Difference (MPD)
Metode MPD dapat menangkap gradasi warna dengan baik dan menggambarkan detail, tetapi akurasi warna tidak terlalu baik.
- Entropy

Entropy mampu untuk menggambar detail, gradasi warna, dan akurasi dengan cukup baik. Hal ini membuat metode Entropy paling efektif digunakan untuk kompresi gambar dengan karakteristik warna yang kompleks.

Kesimpulan

Penggunaan metode kompresi Quadtree efektif untuk gambar yang memiliki banyak area homogen, spada gambar dengan format gambar PNG, gambar berukuran besar/gambar dengan detail yang sebenarnya tidak terlalu penting untuk ditampilkan secara penuh. Hal ini karena Quadtree mampu menyederhanakan area yang seragam menjadi satu blok, sehingga mengurangi ukuran file tanpa mengorbankan informasi penting.

Salah satu alasan metode ini cocok untuk file PNG karena PNG menggunakan *lossless compression*. Semua detail gambar dipertahankan secara utuh tanpa menghapus informasi apapun. Sehingga metode ini cocok untuk menyederhanakannya dengan kompresinya yang hanya berdasarkan warna.

Di sisi lain, metode ini kurang optimal untuk gambar dengan detail sangat tinggi, noise merata, atau gambar berukuran kecil. Pada kondisi tersebut, pembagian blok justru akan lebih banyak, sehingga ukuran file tidak berkurang signifikan atau bahkan merusak kualitas visual gambar.

LAMPIRAN

Repository Github :

https://github.com/salmaanhaniif/Tucil2_13523056.git

Tabel :

No	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 sendiri	√	