

Laporan Tugas Kecil 2
IF2211 Strategi Algoritma
Kompresi Gambar dengan Metode Quadtree



Disusun Oleh
13523115 - Azfa Radhiyya Hakim

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

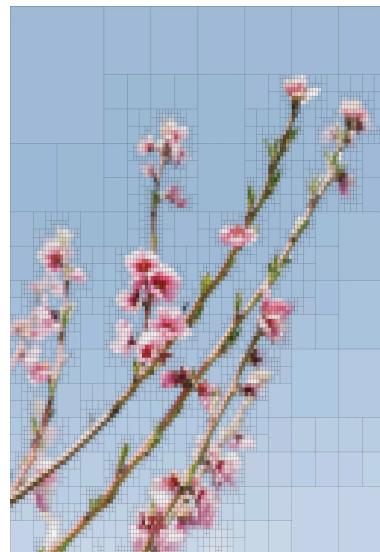
DAFTAR ISI

DAFTAR ISI.....	1
BAB 1	
DESKRIPSI MASALAH DAN ALGORITMA.....	2
1.1. Quadtree.....	2
1.2. Analisis dan Implementasi dengan Algoritma Divide and Conquer.....	3
BAB 2	
STRUKTUR FILE, CLASS, DAN FUNGSI.....	6
2.1. Struktur File.....	6
2.2. Class ImageExtract.....	6
2.3. Class QuadTreeNode.....	7
2.4. File ImageDetection.cpp.....	8
2.5. Class QuadTree.....	8
2.6. File main.cpp.....	10
BAB 3	
SOURCE CODE PROGRAM.....	11
3.1. Repository Github.....	11
3.2. Source Code.....	11
BAB 4	
UJI COBA PROGRAM.....	28
BAB 5	
ANALISIS.....	37
BAB 6	
IMPLEMENTASI BONUS.....	38
LAMPIRAN.....	42

BAB 1

DESKRIPSI MASALAH DAN ALGORITMA

1.1. Quadtree



Gambar 1. Quadtree dalam Kompresi Gambar

(Sumber: <https://medium.com/@tannerwyk/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.

1.2. Analisis dan Implementasi dengan Algoritma Divide and Conquer

Algoritma Divide and Conquer dapat digunakan untuk mengkompresi gambar menggunakan QuadTree. Algoritma dimulai dengan mendefinisikan terlebih dahulu blok pixel yang diperoleh dari gambar ingin dikompresi. Dari data tersebut, kemudian akan diiterasi seluruh blok untuk menentukan nilai error yang dimiliki. Perhitungan error dilakukan dengan lima metode, yaitu

1. Variance

Yaitu dengan mengukur persebaran warna relatif terhadap nilai rata-ratanya.

<u>Variance</u>	$\sigma_c^2 = \frac{1}{N} \sum_{i=1}^N (P_{i,c} - \mu_c)^2$ $\sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$
	σ_c^2 = Variansi tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok

2. Mean Absolute Deviation (MAD)

Yaitu dengan menghitung nilai deviasi rata-rata dari tiap piksel.

Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $ $MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok

3. Max Pixel Difference

Menghitung perbedaan terbesar antara piksel dalam blok

Max Pixel Difference	$D_c = \max(P_{i,c}) - \min(P_{i,c})$ $D_{RGB} = \frac{D_R + D_G + D_B}{3}$
	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c

4. Entropy

Menentukan keragaman informasi warna yang ada pada blok

<u>Entropy</u>	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok $P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

5. Structural Similarity Index (SSIM)

Mengukur kesamaan Luminance, Contrast, dan Structure dari blok gambar dengan nilai rata-ratanya. Untuk 24 bit channel RGB (8 bit per kanal, 0-255), nilai dari $C_1 = (0.01 \cdot 255)^2 = 6.5025$ dan $C_2 = (0.03 \cdot 255)^2 = 58.5225$.

Sedangkan nilai Wr, Wg, dan Wb adalah $\frac{1}{3}$, agar seimbang antar proporsi warna.

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

Nilai dari error ini kemudian akan dibandingkan dengan nilai ambang batas (threshold) yang telah ditentukan. Jika nilai dari error berada di atas threshold, ukuran blok lebih besar dari minimum block size yang telah ditetapkan, dan ukuran blok setelah dibagi menjadi empat tidak kurang dari minimum block size, maka blok ini akan dibagi menjadi empat sub-blok kuadran: kiri atas, kanan atas, kiri bawah, dan kanan bawah. Proses yang sama akan diulang secara rekursif oleh masing-masing sub-blok, membentuk struktur pohon yang merepresentasikan pemecahan gambar secara hierarkis.

Namun jika kondisi diatas tidak terpenuhi, maka blok tidak akan dibagi lagi, dan akan dilakukan normalisasi warna pada blok dan ditetapkan bahwa untuk blok dengan posisi dan ukuran tersebut akan berwarna sama sesuai dengan warna rata-rata. Proses rekursif ini akan terus berlanjut hingga seluruh area gambar telah dievaluasi dan semua blok memenuhi kriteria yang telah ditentukan sebelumnya. Hasil akhirnya adalah sebuah struktur QuadTree yang menyimpan informasi area homogen dalam gambar. Struktur ini kemudian dapat digunakan untuk membangun kembali gambar yang telah terkompresi.

Berikut adalah pseudocode dari algoritma yang dibuat.

```
procedure startDNC(input Gambar : MatriksPixel, threshold: integer, minSize : integer, metode : string, output root : Node)
{ Melakukan kompresi gambar dengan membangun struktur Quadtree menggunakan pendekatan divide and conquer.
  Masukan: Gambar berupa matriks pixel, ambang batas threshold, ukuran blok minimum, dan metode evaluasi homogenitas
  Keluaran: Akar pohon Quadtree yang merepresentasikan hasil kompresi }

Deklarasi:
  root : Node

Algoritma:
  root <- buildQuadTree(Gambar, 0, 0, width(Gambar), height(Gambar), threshold, minSize, metode)
```

```
procedure buildQuadTree(input Gambar : MatriksPixel, x, y, width, height,
threshold, minSize : integer, metode : string, output simpul : Node)
{ Fungsi rekursif untuk membangun Quadtree. Membagi blok jika tidak homogen dan ukurannya > minSize }

Deklarasi:
  err : real
  simpul1, simpul2, simpul3, simpul4 : Node
  midX, midY : integer

Algoritma:
  if width <= minSize or height <= minSize then
    simpul <- makeLeaf(Gambar, x, y, width, height)
  else
    err <- calculateError(Gambar, x, y, width, height, metode)
    if err <= threshold then
      simpul <- makeLeaf(Gambar, x, y, width, height)
    else
      midX <- (x + width) div 2
      midY <- (y + height) div 2

      buildQuadTree(Gambar, x, y, width div 2, height div 2, threshold,
minSize, metode, simpul1)
      buildQuadTree(Gambar, midX, y, width div 2, height div 2, threshold,
minSize, metode, simpul2)
      buildQuadTree(Gambar, x, midY, width div 2, height div 2, threshold,
minSize, metode, simpul3)
      buildQuadTree(Gambar, midX, midY, width div 2, height div 2,
threshold, minSize, metode, simpul4)

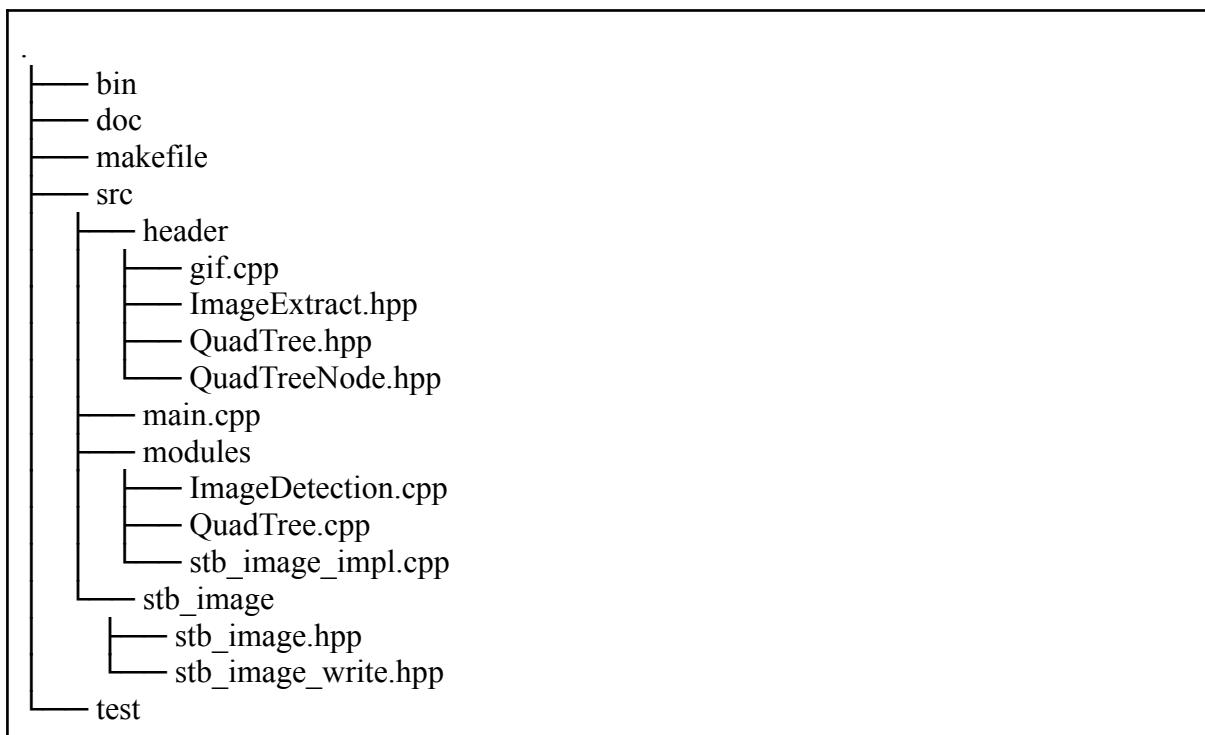
      simpul <- setChildren(simpul1, simpul2, simpul3, simpul4)
    endif
  endif
```

BAB 2

STRUKTUR FILE, CLASS, DAN FUNGSI

2.1. Struktur File

Berikut ini adalah struktur file yang digunakan.



2.2. Class ImageExtract

Atribut	Deskripsi
vector<vector<RGB>> pixels;	Menyimpan data gambar dalam bentuk array 2D, yang berisi informasi terkait proporsi warna Red, Green, dan Blue.
int width	Lebar dari pixels
int height	Tinggi dari pixels
int N	Banyak pixel dalam blok

Metode	Deskripsi
ImageExtract()	Konstruktor class

bool loadImage(const string& filename)	Digunakan untuk mengubah image menjadi vector<vector<RGB>> dengan mengiterasi tiap data. Menghasilkan true jika gambar berhasil di akses.
void saveImage(const vector<vector<RGB>>& imgData, const string& filename)	Digunakan untuk mengkonversi data pixel yang telah diubah menjadi sebuah gambar, menyesuaikan dengan input nama file
vector<vector<RGB>> getPixels()	Mengembalikan atribut pixel dari class
int getWidth()	Mengembalikan atribut width dari class
int getHeight()	Mengembalikan atribut height dari class
int pixelCount()	Menghitung banyak pixel
long getFileSize(const string& filename)	Mengembalikan ukuran dari file

2.3. Class QuadTreeNode

Atribut	Deskripsi
int x	Posisi tinggi pada blok pojok kiri atas
int y	Posisi lebar pada blok pojok kiri atas
int height	Tinggi dari blok
int width	Lebar dari blok
RGB averageColor	RGB yang berisi informasi rata-rata proporsi Red, Green, dan Blue pada blok
bool isLeaf	Ber(nilai true jika node ini adalah sebuah daun
shared_ptr<QuadTreeNode> topLeft; shared_ptr<QuadTreeNode> topRight; shared_ptr<QuadTreeNode> bottomLeft; shared_ptr<QuadTreeNode> bottomRight;	Alamat anak dari node

Metode	Deskripsi
QuadTreeNode(int x, int y, int height, int width, RGB color, bool isLeaf) {}	Konstruktor class

void setChildren(std::shared_ptr<QuadTreeNode> tl, std::shared_ptr<QuadTreeNode> tr, std::shared_ptr<QuadTreeNode> bl, std::shared_ptr<QuadTreeNode> br) {}	Setter untuk keempat anak dari node.
int getX() const {}	Mengambil nilai x
int getY() const {}	Mengambil nilai y
int getHeight() const {}	Mengambil atribut tinggi
int getWidth() const {}	Mengambil atribut Lebar
RGB getAverageColor() const {}	Mengembalikan atribut averageColor
bool getIsLeaf() const {}	BerNilai true jika node adalah sebuah daun
shared_ptr<QuadTreeNode> getTopLeft() const { return topLeft; } shared_ptr<QuadTreeNode> getTopRight() const { return topRight; } shared_ptr<QuadTreeNode> getBottomLeft() const { return bottomLeft; } shared_ptr<QuadTreeNode> getBottomRight() const { return bottomRight; }	Mengembalikan alamat anak dari node.

2.4. File ImageDetection.cpp

ImageDetection.cpp adalah file yang berisi fungsi-fungsi untuk medeteksi jenis file gambar. Metode yang digunakan adalah dengan membaca magic number dari file yang diinput.

Nama Fungsi	Deskripsi
string isImageFile(const string& filePath)	Mengembalikan tipe file dari input file. Jika bukan merupakan JPG, JPEG, atau PNG, maka akan mengembalikan penanda “#”

2.5. Class QuadTree

Atribut	Deskripsi
shared_ptr<QuadTreeNode> root	Menyimpan node akar dari struktur quadtree yang dibentuk.
double threshold	Nilai ambang batas yang digunakan untuk menentukan apakah suatu blok pixel bisa dikompresi.
int minSize	Ukuran minimum blok pixel yang masih bisa dibagi lebih lanjut.
int errorMethod	Metode perhitungan galat (error) yang digunakan untuk menentukan keseragaman blok pixel.

Metode	Deskripsi
QuadTree(double threshold, int minSize, int errorMethod)	Konstruktor class untuk inisialisasi atribut dan membangun struktur quadtree.
RGB calculateAverage(int x, int y, int height, int width)	Menghitung rata-rata warna RGB dari area gambar yang ditentukan.
double errorVariance(int x, int y, int height, int width)	Menghitung galat berdasarkan variansi nilai RGB pada area tertentu.
double errorMAD(int x, int y, int height, int width)	Menghitung galat berdasarkan Mean Absolute Deviation (MAD).
double errorMaxPixelDifference(int x, int y, int height, int width)	Menghitung galat berdasarkan selisih maksimum antar piksel.
double errorEntropy(int x, int y, int height, int width)	Menghitung galat berdasarkan entropi dari distribusi warna di area tertentu.
double errorSSIM(int x, int y, int height, int width)	Menghitung galat menggunakan metode Structural Similarity Index (SSIM).
shared_ptr<QuadTreeNode> makeLeaf(int x, int y, int width, int height)	Membuat leaf node ketika area piksel dianggap seragam atau terlalu kecil untuk dibagi.
shared_ptr<QuadTreeNode> buildQuadTree(int x, int y, int width, int height, double threshold, int minSize, int errorMethod)	Membangun quadtree secara rekursif dengan membagi area piksel menjadi 4 bagian jika perlu.

void startDNC(const vector<vector<RGB>>& originalImage)	Memulai proses divide and conquer dari data gambar asli.
vector<vector<RGB>> reconstructImage(int width, int height)	Menghasilkan gambar hasil kompresi dari struktur quadtree.
void constructImageRec(shared_ptr<QuadTreeNode> node, vector<vector<RGB>>& result)	Metode rekursif untuk merekonstruksi ulang gambar dari node-node quadtree.
int countNodes(shared_ptr<QuadTreeNode> node) const	Menghitung total node (termasuk leaf dan non-leaf) dari sebuah node tertentu.
int totalNodes() const	Mengembalikan total jumlah node pada quadtree dari root.
int countLeafNodes(shared_ptr<QuadTreeNode> node) const	Menghitung jumlah leaf node dari sebuah node tertentu.
int leafNodes() const	Mengembalikan jumlah total leaf node pada quadtree.
int countDepth(shared_ptr<QuadTreeNode> node) const	Menghitung kedalaman dari quadtree yang dimulai dari node tertentu.
int depth() const	Mengembalikan kedalaman total dari quadtree.
void drawQuadTreeBoundaries(vector<vector<RGB>>& image, shared_ptr<QuadTreeNode> node, const RGB& boundaryColor)	Menggambar batas antara blok-blok piksel yang seragam sebagai visualisasi quadtree.
vector<vector<RGB>> getQuadTreeVisualization(int width, int height)	Menghasilkan visualisasi quadtree berupa gambar dengan batas-batas blok terkompresi.
shared_ptr<QuadTreeNode> getRoot() const	Mengembalikan pointer ke node root dari quadtree.
void createCompressionGIF(const std::string& outputPath, int width, int height)	Membuat GIF yang menunjukkan proses kompresi gambar secara bertahap.

2.6. File main.cpp

File ini adalah file yang akan di run untuk menjalankan program, yang akan mengkonstruksi class dan memanggil method yang dimiliki masing-masing class.

BAB 3

SOURCE CODE PROGRAM

3.1. Repository Github

Link Github: https://github.com/azfaradhi/Tucil2_13523115

3.2. Source Code

Bahasa yang digunakan: C++

1. ImageExtract.hpp

```
/*
- Class yang digunakan untuk mengekstrak gambar dari input file
- Disini dibuat struct RGB untuk menyimpan data warna RGB dari pixel yang didapat
- Class ImageExtract memiliki atribut pixel (vector dari vector RGB), lebar,
tinggi, dan jumlah pixel
- Fungsi stbi_load dan stbi_write didapat dari library stb_image.h dan
stb_image_write.h

*/
#ifndef IMAGE_EXTRACT_HPP
#define IMAGE_EXTRACT_HPP
#include <iostream>
#include <vector>
#include <fstream>

#include "../stb_image/stb_image.hpp"
#include "../stb_image/stb_image_write.hpp"

using namespace std;

struct RGB {
    double r, g, b;
    RGB() : r(0), g(0), b(0) {}
    RGB(double r, double g, double b) : r(r), g(g), b(b) {}
};

class ImageExtract {
private:
    vector<vector<RGB>> pixels;
    int width;
    int height;
    int N;
public:
    ImageExtract() : width(0), height(0), N(0) {}

    ImageExtract(int height, int width, int N, vector<vector<RGB>> pixels) {
        this->height = height;
        this->width = width;
        this->N = N;
        this->pixels = pixels;
    }

    ~ImageExtract() {}
}
```

```

        bool loadImage(const string& filename) {
            int channels;
            unsigned char* data = stbi_load(filename.c_str(), &width, &height,
&channels, 3);
            N = width * height;
            if (!data) {
                return false;
            }

            pixels.resize(height);
            for (int i = 0; i < height; i++) {
                pixels[i].resize(width);
                for (int j = 0; j < width; j++) {
                    int pos = (i * width + j) * 3;
                    pixels[i][j] = RGB(data[pos], data[pos + 1], data[pos + 2]);
                }
            }
            stbi_image_free(data);
            return true;
        }

void saveImage(const vector<vector<RGB>>& imgData, const string& filename) {
    int height = imgData.size();
    int width = imgData[0].size();
    vector<unsigned char> data(width * height * 3);

    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            int idx = (i * width + j) * 3;
            data[idx] = imgData[i][j].r;
            data[idx + 1] = imgData[i][j].g;
            data[idx + 2] = imgData[i][j].b;
        }
    }
    stbi_write_jpg(filename.c_str(), width, height, 3, data.data(), 70);
}

vector<vector<RGB>> getPixels() const { return pixels; }
int getWidth() const { return width; }
int getHeight() const { return height; }
int pixelCount() const { return N; }
long getFileSize(const string& filename) {
    ifstream file(filename, ios::binary | ios::ate);
    return file.tellg();
}
};

#endif

```

2. QuadTreeNode.hpp

```

/*
- Class QuadTreeNode digunakan sebagai pondasi utama dari algoritama DNC
- Memiliki atribut x,y sebagai posisi kiri atas dari node terhadap pixel,
- Memiliki atribut height dan width sebagai ukuran dari node yang sudah dibagi
- Memiliki atribut averageColor untuk menyimpan warna rata-rata dari node
tersebut, diisi ketika node yang bersangkutan adalah daun
- Memiliki atribut isLeaf sebagai penanda jika node adalah sebuah daun

```

```

- shared_ptr<QuadTreeNode> topLeft, topRight, bottomLeft, bottomRight untuk
menyimpan anak dari node tersebut
*/



#ifndef _QUADTREE_NODE_HPP_
#define _QUADTREE_NODE_HPP_


#include <iostream>
#include <memory>
#include "ImageExtract.hpp"

class QuadTreeNode {
private:
    int x,y,height,width;
    RGB averageColor;
    bool isLeaf;
    shared_ptr<QuadTreeNode> topLeft;
    shared_ptr<QuadTreeNode> topRight;
    shared_ptr<QuadTreeNode> bottomLeft;
    shared_ptr<QuadTreeNode> bottomRight;
public:
    QuadTreeNode(int x, int y, int height, int width, RGB color, bool isLeaf) {
        this->x = x;
        this->y = y;
        this->height = height;
        this->width = width;
        this->averageColor = color;
        this->isLeaf = isLeaf;
        this->topLeft = nullptr;
        this->topRight = nullptr;
        this->bottomLeft = nullptr;
        this->bottomRight = nullptr;
    }

    ~QuadTreeNode() {};

    void setChildren(
        std::shared_ptr<QuadTreeNode> tl,
        std::shared_ptr<QuadTreeNode> tr,
        std::shared_ptr<QuadTreeNode> bl,
        std::shared_ptr<QuadTreeNode> br
    )
    {
        topLeft = tl;
        topRight = tr;
        bottomLeft = bl;
        bottomRight = br;
        isLeaf = false;
    }

    // Metode getter
    int getX() const { return x; }
    int getY() const { return y; }
    int getHeight() const { return height; }
    int getWidth() const { return width; }
    RGB getAverageColor() const { return averageColor; }
    bool getIsLeaf() const { return isLeaf; }

    // Mengambil node dari anak
    shared_ptr<QuadTreeNode> getTopLeft() const { return topLeft; }
    shared_ptr<QuadTreeNode> getTopRight() const { return topRight; }
    shared_ptr<QuadTreeNode> getBottomLeft() const { return bottomLeft; }
    shared_ptr<QuadTreeNode> getBottomRight() const { return bottomRight; }
};


```

```
#endif
```

3. ImageDetection.cpp

```
// ImageDetection.cpp adalah file yang berisi fungsi-fungsi untuk mendekripsi jenis
// file gambar.
// Metode yang digunakan adalah dengan membaca magic number dari file yang
// diinput.

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

string isImageFile(const string& filePath) {
    ifstream file(filePath, ios::binary);
    if (!file) {
        cerr << "Gagal membuka file: " << filePath << endl;
        return "#";
    }

    // Pengecekan magic number yaitu dengan membaca 8 byte pertama dari file
    vector<unsigned char> buffer(8);
    file.read(reinterpret_cast<char*>(buffer.data()), buffer.size());

    if (buffer.size() < 3) return "#";

    if (buffer[0] == 0xFF && buffer[1] == 0xD8 && buffer[2] == 0xFF) {
        return "JPG";
    }

    if (buffer.size() >= 8 && buffer[0] == 0x89 && buffer[1] == 0x50 &&
        buffer[2] == 0x4E && buffer[3] == 0x47 && buffer[4] == 0x0D &&
        buffer[5] == 0x0A && buffer[6] == 0x1A && buffer[7] == 0x0A) {
        return "PNG";
    }
    cout << "File yang diberikan bukan gambar!" << endl;
    return "#";
}
```

4. QuadTree.hpp

```
/*
- Class QuadTree digunakan untuk menyimpan quadtree yang telah dibuat
- Memiliki atribut root, sebagai alamat awal dari quadtree yang dibuat
- Memiliki atribut threshold, minSize, errorMethod untuk menyimpan parameter yang
digunakan dalam algoritma DNC
- Atribut pixels digunakan untuk menyimpan data pixel yang didapat dari gambar
asli
*/
#ifndef _QUADTREE_HPP_
#define _QUADTREE_HPP_

#include <iostream>
#include <memory>
#include <unordered_map>
#include <math.h>
```

```

#include <queue>

#include "ImageExtract.hpp"
#include "QuadTreeNode.hpp"

using namespace std;

class QuadTree{
private:
    shared_ptr<QuadTreeNode> root;
    double threshold;
    int minSize;
    int errorMethod;
    vector<vector<RGB>> pixels;

public:
    QuadTree(double threshold, int minSize, int errorMethod);
    // Ini adalah konstruktor untuk membangun quadtree

    RGB calculateAverage(int x, int y, int height, int width);
    // Metode yang digunakan untuk menghitung rata-rata warna dari tiap elemen
    RGB pada bagian pixel yang bersesuaian

    double errorVariance(int x, int y, int height, int width);
    double errorMAD(int x, int y, int height, int width);
    double errorMaxPixelDifference(int x, int y, int height, int width);
    double errorEntropy(int x, int y, int height, int width);
    double errorSSIM(int x, int y, int height, int width);
    // Lima metode error yang berbeda berdasarkan parameter threshold dan minSize
    yang diberikan

    shared_ptr<QuadTreeNode> makeLeaf(int x, int y, int width, int height);
    // Dipanggil ketika suatu node sudah dapat dibagi berdasarkan aturan
    yang berlaku

    shared_ptr<QuadTreeNode> buildQuadTree(int x, int y, int width, int height,
    double threshold, int minSize, int errorMethod);
    // Jika node masih dapat dibagi, maka akan membagi node tersebut menjadi 4
    bagian, yaitu topLeft, topRight, bottomLeft, dan bottomRight

    void startDNC(const vector<vector<RGB>>& originalImage);
    // Fungsi yang dipanggil pada main dan sebagai konstruktor pixels dari class

    vector<vector<RGB>> reconstructImage(int width, int height);
    // Fungsi yang dipanggil pada main untuk merekonstruksi ulang gambar hasil
    kompresi

    void constructImageRec(shared_ptr<QuadTreeNode> node, vector<vector<RGB>>&
    result);
    // Fungsi rekursif yang digunakan untuk mengisi gambar hasil kompresi

    int countNodes(shared_ptr<QuadTreeNode> node) const;
    int totalNodes() const;
    // Menghitung jumlah nodes dalam quadtree

    int countLeafNodes(shared_ptr<QuadTreeNode> node) const;
    int leafNodes() const;
    // Menghitung jumlah node daun dalam quadtree

    int countDepth(shared_ptr<QuadTreeNode> node) const;
    int depth() const;
    // Menghitung kedalaman quadtree

```

```

shared_ptr<QuadTreeNode> getRoot() const { return root; }
// Mengambil alamat root

void createCompressionGIF(const std::string& outputPath, int width, int
height);
    // membuat gif hasil kompresi
};

#endif

```

5. Quadtree.cpp

```

#include "../header/QuadTree.hpp"
#include "../header/gif.hpp"

QuadTree::QuadTree(double threshold, int minSize, int errorMethod) {
    this->root = nullptr;
    this->threshold = threshold;
    this->minSize = minSize;
    this->errorMethod = errorMethod;
}

RGB QuadTree::calculateAverage(int x, int y, int height, int width){
    double avgR = 0;
    double avgB = 0;
    double avgG = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            avgR += pixels[i][j].r;
            avgG += pixels[i][j].g;
            avgB += pixels[i][j].b;
        }
    }
    return RGB(avgR/(height*width), avgG/(height*width), avgB/(height*width));
}

double QuadTree::errorVariance(int x, int y, int height, int width) {

    RGB avg = calculateAverage(x, y, height, width);

    double variance = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            variance += ((pixels[i][j].r - avg.r) * (pixels[i][j].r - avg.r)) +
((pixels[i][j].g - avg.g) * (pixels[i][j].g - avg.g)) + ((pixels[i][j].b - avg.b)
* (pixels[i][j].b - avg.b));
        }
    }
    return variance / (height * width * 3);
}

double QuadTree::errorMAD(int x, int y, int height, int width) {

    RGB avg = calculateAverage(x, y, height, width);

    double mad = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            mad += abs(pixels[i][j].r - avg.r) + abs(pixels[i][j].g - avg.g) +

```

```

abs(pixels[i][j].b - avg.b);
    }
}
return mad / (height * width * 3);
}

double QuadTree::errorMaxPixelDifference(int x, int y, int height, int width) {
    unsigned char minR = 255, minG = 255, minB = 255;
    unsigned char maxR = 0, maxG = 0, maxB = 0;

    for (int i = y; i < y + height; ++i) {
        for (int j = x; j < x + width; ++j) {
            if (pixels[i][j].r < minR) minR = pixels[i][j].r;
            if (pixels[i][j].g < minG) minG = pixels[i][j].g;
            if (pixels[i][j].b < minB) minB = pixels[i][j].b;
            if (pixels[i][j].r > maxR) maxR = pixels[i][j].r;
            if (pixels[i][j].g > maxG) maxG = pixels[i][j].g;
            if (pixels[i][j].b > maxB) maxB = pixels[i][j].b;
        }
    }
    return ((maxR - minR) + (maxG - minG) + (maxB - minB)) / 3;
}

double QuadTree::errorEntropy(int x, int y, int height, int width) {
    unordered_map<int,int> freqR, freqG, freqB;
    const int total = width * height;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            freqR[pixels[i][j].r]++;
            freqG[pixels[i][j].g]++;
            freqB[pixels[i][j].b]++;
        }
    }

    double entropyR = 0.0;
    double entropyG = 0.0;
    double entropyB = 0.0;

    for (const auto& pair : freqR) {
        double p = (double)pair.second / total;
        entropyR -= p * log2(p);
    }
    for (const auto& pair : freqG) {
        double p = (double)pair.second / total;
        entropyG -= p * log2(p);
    }
    for (const auto& pair : freqB) {
        double p = (double)pair.second / total;
        entropyB -= p * log2(p);
    }

    return (entropyR + entropyG + entropyB) / 3;
}

double QuadTree::errorSSIM(int x, int y, int height, int width) {

    RGB avg = calculateAverage(x, y, height, width);
    double uRx = avg.r;
    double uGx = avg.g;
    double uBx = avg.b;
    double uRy = avg.r;
    double uGy = avg.g;
    double uBy = avg.b;

    double sigmaRy = 0;

```

```

double sigmaGy = 0;
double sigmaBy = 0;
double c1 = 6.5025;
double c2 = 58.5225;

double sigmaRx = 0;
double sigmaGx = 0;
double sigmaBx = 0;

for (int i = y; i < y + height; i++){
    for (int j = x; j < x+width; j++){
        sigmaRx += (pixels[i][j].r - uRx) * (pixels[i][j].r - uRx);
        sigmaGx += (pixels[i][j].g - uGx) * (pixels[i][j].g - uGx);
        sigmaBx += (pixels[i][j].b - uBx) * (pixels[i][j].b - uBx);
    }
}
sigmaRx /= (height * width);
sigmaGx /= (height * width);
sigmaBx /= (height * width);

double ssimr = ((2 * uRx * uRy + c1) * c2) / (((uRx * uRx) + (uRy * uRy) +
c1) * (sigmaRx + c2));
double ssimg = ((2 * uGx * uGy + c1) * c2) / (((uGx * uGx) + (uGy * uGy) +
c1) * (sigmaGx + c2));
double ssimb = ((2 * uBx * uBy + c1) * c2) / (((uBx * uBx) + (uBy * uBy) +
c1) * (sigmaBx + c2));

return (ssimr + ssimg + ssimb) / 3;
}

shared_ptr<QuadTreeNode> QuadTree::makeLeaf(int x, int y, int height, int width)
{
    return make_shared<QuadTreeNode>(x, y, height, width,
calculateAverage(x,y,height,width), true);
}

shared_ptr<QuadTreeNode> QuadTree::buildQuadTree(int x, int y, int width, int
height, double threshold, int minSize, int errorMethod) {

    if (width * height <= minSize){
        return makeLeaf(x,y,height,width);
    }

    double error = 0;
    if (errorMethod == 1){
        error = errorVariance(x,y,height,width);
    }
    else if (errorMethod == 2){
        error = errorMAD(x,y,height,width);
    }
    else if (errorMethod == 3){
        error = errorMaxPixelDifference(x,y,height,width);
    }
    else if (errorMethod == 4){
        error = errorEntropy(x,y,height,width);
    }
    else if (errorMethod == 5){
        error = errorSSIM(x,y,height,width);
    }
    else{
        cout << "Error: Metode error tidak valid!" << endl;
        return nullptr;
    }

    if (errorMethod == 5){
        if (error >= threshold){
            return makeLeaf(x,y,height,width);
        }
    }
}

```

```

        }
    }
    else {
        if (error <= threshold){
            return makeLeaf(x,y,height,width);
        }
    }

    int halfW = width/2;
    int resW = width % 2;
    int halfH = height/2;
    int resH = height % 2;

    if ((halfW * halfH) < minSize || (halfW + resW) * (halfH + resH) < minSize ||
(halfW + resW) * halfH < minSize || halfW * (halfH + resH) < minSize){
        return makeLeaf(x,y,height,width);
    }
    shared_ptr<QuadTreeNode> tl = buildQuadTree(x, y, halfW, halfH, threshold,
minSize, errorMethod);
    shared_ptr<QuadTreeNode> tr = buildQuadTree(x + halfW, y, halfW + resW,
halfH, threshold, minSize, errorMethod);
    shared_ptr<QuadTreeNode> bl = buildQuadTree(x, y + halfH, halfW, halfH + resH,
threshold, minSize, errorMethod);
    shared_ptr<QuadTreeNode> br = buildQuadTree(x + halfW, y + halfH, halfW + resW,
halfH + resH, threshold, minSize, errorMethod);

    shared_ptr<QuadTreeNode> node = make_shared<QuadTreeNode>(x, y, height,
width, calculateAverage(x,y,height,width), false);
    node->setChildren(tl, tr, bl, br);
    return node;
}

void QuadTree::startDNC(const vector<vector<RGB>>& img) {

    this->pixels = img;
    int height = img.size();
    int width = img[0].size();

    this->root = buildQuadTree(0, 0, width, height, threshold, minSize,
errorMethod);
}

vector<vector<RGB>> QuadTree::reconstructImage(int width, int height) {
    vector<vector<RGB>> result(height, vector<RGB>(width));
    if (root) {
        constructImageRec(root, result);
    }
    return result;
}

void QuadTree::constructImageRec(shared_ptr<QuadTreeNode> node,
vector<vector<RGB>>& result) {
    if (!node) {
        return;
    }

    if (node->getIsLeaf()) {
        for (int j = node->getY(); j < min(node->getY() + node-&gtgetHeight(),
(int)result.size()); j++) {
            for (int i = node->getX(); i < min(node->getX() + node-&gtgetWidth(),
(int)result[0].size()); i++) {
                result[j][i] = node->getAverageColor();
            }
        }
    } else {
        constructImageRec(node->getTopLeft(), result);
        constructImageRec(node->getTopRight(), result);
    }
}

```

```

        constructImageRec(node->getBottomLeft(), result);
        constructImageRec(node->getBottomRight(), result);
    }
}

int QuadTree::countNodes(shared_ptr<QuadTreeNode> node) const {
    if (!node) return 0;

    if (node->getIsLeaf()) {
        return 1;
    }

    return 1 + countNodes(node->getTopLeft()) + countNodes(node->getTopRight()) +
    countNodes(node->getBottomLeft()) + countNodes(node->getBottomRight());
}

int QuadTree::totalNodes() const {
    return countNodes(root);
}

int QuadTree::countLeafNodes(shared_ptr<QuadTreeNode> node) const {
    if (!node) return 0;

    if (node->getIsLeaf()) {
        return 1;
    }

    return countLeafNodes(node->getTopLeft()) +
    countLeafNodes(node->getTopRight()) + countLeafNodes(node->getBottomLeft()) +
    countLeafNodes(node->getBottomRight());
}

int QuadTree::leafNodes() const {
    return countLeafNodes(root);
}

int QuadTree::countDepth(shared_ptr<QuadTreeNode> node) const {
    if (!node) {
        return 0;
    }
    if (node->getIsLeaf()) {
        return 1;
    }

    return 1 + max(countDepth(node->getTopLeft()), max(countDepth(node->getTopRight()), max(countDepth(node->getBottomLeft()), countDepth(node->getBottomRight()))));
}

int QuadTree::depth() const {
    return countDepth(root);
}

void QuadTree::createCompressionGIF(const std::string& outputPath, int width, int height) {

    int imgWidth = width;
    int imgHeight = height;

    GifWriter g;
    GifBegin(&g, outputPath.c_str(), imgWidth, imgHeight, 50);

    std::queue<std::shared_ptr<QuadTreeNode>> nodeQueue;
    std::vector<std::shared_ptr<QuadTreeNode>> allNodes;

    if (root) {

```

```

        nodeQueue.push(root);
    }

    std::vector<uint8_t> originalFrame(imgWidth * imgHeight * 4);
    for (int y = 0; y < imgHeight; y++) {
        for (int x = 0; x < imgWidth; x++) {
            int idx = (y * imgWidth + x) * 4;
            originalFrame[idx] = pixels[y][x].r;
            originalFrame[idx + 1] = pixels[y][x].g;
            originalFrame[idx + 2] = pixels[y][x].b;
        }
    }
    GifWriteFrame(&g, originalFrame.data(), imgWidth, imgHeight, 50);

    std::vector<std::vector<std::shared_ptr<QuadTreeNode>>> levelNodes;

    while (!nodeQueue.empty()) {
        int levelSize = nodeQueue.size();
        std::vector<std::shared_ptr<QuadTreeNode>> currentLevel;

        for (int i = 0; i < levelSize; i++) {
            auto node = nodeQueue.front();
            nodeQueue.pop();
            currentLevel.push_back(node);
            allNodes.push_back(node);

            if (!node->getIsLeaf()) {
                if (node->getTopLeft()) nodeQueue.push(node->getTopLeft());
                if (node->getTopRight()) nodeQueue.push(node->getTopRight());
                if (node->getBottomLeft()) nodeQueue.push(node->getBottomLeft());
                if (node->getBottomRight())
                    nodeQueue.push(node->getBottomRight());
            }
        }

        if (!currentLevel.empty()) {
            levelNodes.push_back(currentLevel);
        }
    }

    for (size_t level = 0; level < levelNodes.size(); level++) {
        std::vector<std::vector<RGB>> reconstruction = pixels;
        for (size_t l = 0; l <= level; l++) {
            for (auto& node : levelNodes[l]) {
                if (node->getIsLeaf()) {
                    for (int y = node->getY(); y < std::min(node->getY() +
node->getHeight(), imgHeight); y++) {
                        for (int x = node->getX(); x < std::min(node->getX() +
node->getWidth(), imgWidth); x++) {
                            reconstruction[y][x] = node->getAverageColor();
                        }
                    }
                }
            }
        }

        std::vector<uint8_t> frame(imgWidth * imgHeight * 4);
        for (int y = 0; y < imgHeight) {
            for (int x = 0; x < imgWidth; x++) {
                int idx = (y * imgWidth + x) * 4;
                frame[idx] = reconstruction[y][x].r;
                frame[idx + 1] = reconstruction[y][x].g;
                frame[idx + 2] = reconstruction[y][x].b;
                frame[idx + 3] = 255;
            }
        }
    }
}

```

```

        GifWriteFrame(&g, frame.data(), imgWidth, imgHeight, 50);
    }

    std::vector<std::vector<RGB>> finalReconstruction =
reconstructImage(imgWidth, imgHeight);
    std::vector<uint8_t> finalFrame(imgWidth * imgHeight * 4);
    for (int y = 0; y < imgHeight; y++) {
        for (int x = 0; x < imgWidth; x++) {
            int idx = (y * imgWidth + x) * 4;
            finalFrame[idx] = finalReconstruction[y][x].r;
            finalFrame[idx + 1] = finalReconstruction[y][x].g;
            finalFrame[idx + 2] = finalReconstruction[y][x].b;
            finalFrame[idx + 3] = 255;
        }
    }

    GifWriteFrame(&g, finalFrame.data(), imgWidth, imgHeight, 50);

    GifEnd(&g);
}

```

6. Main.cpp

```

if (!img.loadImage(filename)) {
    cout << "Gagal load image!" << endl;
    return 1;
}

long orginalSize = img.getFileSize(filename);
int height = img.getHeight();
int width = img.getWidth();
int N = img.pixelCount();
vector<vector<RGB>> pixels = img.getPixels();

int method;
while (true) {
    cout << endl;
    cout << "Masukkan metode yang diinginkan!" << endl;
    cout << "Catatan: 1 = Variance, 2 = MAD, 3 = Max Pixel Difference, 4 =
Entropi, 5 = SSIM" << endl;
    printf("\x1B[32m> \033[0m");
    if (!(cin >> method)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input harus berupa angka! Silahkan coba lagi!" << endl;
        continue;
    }
    if (method < 1 || method > 5) {
        cout << "Metode tidak valid! Silahkan coba lagi!" << endl;
        continue;
    } else {
        break;
    }
}
double threshold;

while (true) {
    cout << endl;
    cout << "Masukkan threshold yang diinginkan!" << endl;
    cout << "Catatan:" << endl;
    cout << "Variance (0 - 16256.25)" << endl;
    cout << "MAD (0 - 127.5)" << endl;
    cout << "Max Pixel Difference (0 - 255)" << endl;
    cout << "Entropi (0 - 8)" << endl;
    cout << "SSIM (0 - 1)" << endl;
    printf("\x1B[32m> \033[0m");
    if (!(cin >> threshold)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input harus berupa double! Silahkan coba lagi!" << endl;
        continue;
    }
    if (method == 1) {
        if (threshold < 0 || threshold > 16256.25) {
            cout << "Range dari threshold tidak valid! Silahkan coba lagi!"
<< endl;
            continue;
        } else {
            break;
        }
    }
    else if (method == 2) {
        if (threshold < 0 || threshold > 127.5) {
            cout << "Range dari threshold tidak valid! Silahkan coba lagi!"
<< endl;
            continue;
        } else {
            break;
        }
    }
}

```

```

    }
    else if (method == 3) {
        if (threshold < 0 || threshold > 255) {
            cout << "Range dari threshold tidak valid! Silahkan coba lagi!"
<< endl;
            continue;
        } else {
            break;
        }
    }
    else if (method == 4) {
        if (threshold < 0 || threshold > 8) {
            cout << "Range dari threshold tidak valid! Silahkan coba lagi!"
<< endl;
            continue;
        } else {
            break;
        }
    }
    else if (method == 5) {
        if (threshold < 0 || threshold > 1) {
            cout << "Range dari threshold tidak valid! Silahkan coba lagi!"
<< endl;
            continue;
        } else {
            break;
        }
    }
}
int minSize;
while (true){
    cout << endl;
    cout << "Masukkan minimum size yang diinginkan!" << endl;
    cout << "Catatan: size tidak boleh lebih besar dari ukuran foto asli!" <<
endl;
    printf("\x1B[32m> \033[0m");
    if (!(cin >> minSize)) {
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input harus berupa angka! Silahkan coba lagi!" << endl;
        continue;
    }
    if (minSize > height * width) {
        cout << "Minimum size tidak valid! Silahkan coba lagi!" << endl;
        continue;
    } else {
        break;
    }
}
double target;
while (true){
    cout << endl;
    cout << "Masukkan target persentase kompresi!" << endl;
    cout << "Catatan: threshold akan disesuaikan jika mengisi ini, jika tetap
ingin menggunakan threshold ketik 0." << endl;
    printf("\x1B[32m> \033[0m");
    if (!(cin >> target)){
        cin.clear();
        cin.ignore(numeric_limits<streamsize>::max(), '\n');
        cout << "Input harus berupa angka! Silahkan coba lagi!" << endl;
        continue;
    }
    if (target == 0){
        isTarget = false;
    }
    else{
        isTarget = true;
    }
}

```

```

        }
        break;
    }
    string outputFileName;
    string outputGifName;
    cout << endl;
    cout << "Masukkan alamat relatif untuk menyimpan gambar! (Contoh:  

test/pohon_output.jpg)" << endl;
    printf("\x1B[32m>> \033[0m");
    cin >> outputFileName;
    cout << endl;
    cout << "Masukkan alamt relatif untuk menyimpan GIF! (Contoh:  

test/pohon_output.gif)" << endl;
    printf("\x1B[32m>> \033[0m");
    cin >> outputGifName;

    size_t lastSlash = outputGifName.find_last_of("/\\");
    if (lastSlash != string::npos) {
        string directory = outputGifName.substr(0, lastSlash);
        std::filesystem::create_directories(directory);
    }
    lastSlash = outputFileName.find_last_of("/\\");
    if (lastSlash != string::npos) {
        string directory = outputFileName.substr(0, lastSlash);
        std::filesystem::create_directories(directory);
    }

    if (target == 0){
        thershouldNow = threshold;
    }
    else{
        if (method == 1){
            thershouldMax = 16256.25;
            thershouldMin = 0;
        }
        else if (method == 2){
            thershouldMax = 127.5;
            thershouldMin = 0;
        }
        else if (method == 3){
            thershouldMax = 255;
            thershouldMin = 0;
        }
        else if (method == 4){
            thershouldMax = 8;
            thershouldMin = 0;
        }
        else if (method == 5){
            thershouldMax = 1;
            thershouldMin = 0;
        }
        thershouldNow = (thershouldMax + thershouldMin) / 2;
    }

    cout << "Memulai kompresi ..." << endl;
    int totalnodes = 0;
    int leafnodes = 0;
    int depth = 0;
    double ratio = 0;
    long newSize = 0;
    auto startCompression = high_resolution_clock::now();
    while (true){
        QuadTree quadtree(thershouldNow, minSize, method);
        quadtree.startDNC(pixels);
        vector<vector<RGB>> reconstructedImage = quadtree.reconstructImage(width,
height);

```

```

        img.saveImage(reconstructedImage, outputFileName);
        newSize = img.getFileSize(outputFileName);

        if (isTarget){
            ratio = (double) (1 - (double)newSize/orginalSize)* 100;
            if (abs(ratio - target) < 0.01){
                cout << "Berhasil mencapai target dengan selisih 0.01!" << endl;
                totalnodes = quadtree.totalNodes();
                leafnodes = quadtree.leafNodes();
                depth = quadtree.depth();
                cout << "Membuat GIF kompresi..." << endl;
                quadtree.createCompressionGIF(outputGifName , width, height);
                break;
            }
            else if (abs(thersholdMax - thersholtMin) < 0.01){
                cout << "Threshold tidak dapat diturunkan lagi!!" << endl;
                cout << "Berhasil mencapai target!" << endl;
                totalnodes = quadtree.totalNodes();
                leafnodes = quadtree.leafNodes();
                depth = quadtree.depth();
                cout << "Membuat GIF kompresi..." << endl;
                quadtree.createCompressionGIF(outputGifName , width, height);
                break;
            }
        }

        if (ratio < target){
            if (method == 5){
                thersholtMax = thersholtNow;
                thersholtNow = (thersholdMax + thersholtMin) / 2;
            }
            else{
                thersholtMin = thersholtNow;
                thersholtNow = (thersholdMax + thersholtMin) / 2;
            }
        }
        else if (ratio > target){
            if (method == 5){
                thersholtMin = thersholtNow;
                thersholtNow = (thersholdMax + thersholtMin) / 2;
            }
            else{
                thersholtMax = thersholtNow;
                thersholtNow = (thersholdMax + thersholtMin) / 2;
            }
        }
        cout << "Persentase kompresi saat ini: " << ratio << "%" << endl;
        remove(outputFileName.c_str()));

    }
    else{
        cout << "Berhasil mengkompresi gambar!" << endl;
        ratio = (double) (1 - (double)newSize/orginalSize)* 100;
        totalnodes = quadtree.totalNodes();
        leafnodes = quadtree.leafNodes();
        depth = quadtree.depth();
        cout << "Membuat GIF kompresi..." << endl;
        quadtree.createCompressionGIF(outputGifName, width, height);
        break;
    }
}

auto endCompression = high_resolution_clock::now();
auto compressionTime = duration_cast<milliseconds>(endCompression -
startCompression).count();

cout << "Ukuran file asli: " << orginalSize << " bytes" << endl;

```

```
cout << "Ukuran file baru: " << newSize << " bytes" << endl;
cout << "Waktu kompresi: " << compressionTime << " ms" << endl;
cout << "Rasio kompresi: " << ratio << "%" << endl;
cout << "Total node: " << totalnodes << endl;
cout << "Node daun: " << leafnodes << endl;
cout << "Kedalaman: " << depth << endl;

cout << "Gambar dan GIF telah berhasil disimpan!" << endl;
cout << "(Catatan: kadang file hasil kompresi tidak langsung muncul pada
vscode, silahkan cek pada explorer)" << endl;

return 0;
}
```

BAB 4

UJI COBA PROGRAM

Test Case #1	
Input	
Metode	Variance
Threshold	50
Minimum Size	64
Output data	<pre>Original file size: 75301 bytes New file size: 50962 bytes Compression time: 3761 ms Compression ratio: 32.3223% Total nodes: 2601 Leaf nodes: 1951 Depth: 7</pre> <p>Ukuran file asli: 75301 bytes Ukuran file baru: 50962 bytes Waktu kompresi: 3761 ms Rasio kompresi: 32.3223% Total node: 2601 Node daun: 1951 Kedalaman: 7</p>

Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/mrauraVariance.gif
Test Case #2	
Input	
Metode	Max Pixel Difference
Threshold	50
Minimum Size	64
Output data	Ukuran file asli: 1327528bytes Ukuran file baru: 49884 bytes Waktu kompresi: 3667 ms Rasio kompresi: 96.2423 % Total node: 3533 Node daun: 2650 Kedalaman: 7

Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/mountainDifference.gif
Test Case #3	
Input	
Metode	Mean Absolute Deviation (MAD)
Threshold	4
Minimum Size	16
Output data	Ukuran file asli: 41856 bytes Ukuran file baru: 25217 bytes Waktu kompresi: 1548 ms Rasio kompresi: 39.753% Total node: 7973 Node daun: 5980 Kedalaman: 9

Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/penaconomyMAD.gif
Test Case #4	
Input	
Metode	Variance
Threshold	8
Minimum Size	6
Output data	Ukuran file asli: 65015 bytes Ukuran file baru: 55429 bytes Waktu kompresi: 3645 ms Rasio kompresi: 14.7443% Total node: 47957 Node daun: 35968 Kedalaman: 9

Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/spidermanVariance.gif
Test Case #5	
Input	
Metode	SSIM
Threshold	0.8
Minimum Size	16
Output data	Ukuran file asli: 184000 bytes Ukuran file baru: 122929 bytes Waktu kompresi: 6679 ms Rasio kompresi: 33.1908% Total node: 40309 Node daun: 30232 Kedalaman: 9

Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/on epieceSSIM.gif
Test Case #6	
Input	
Metode	Variance
Threshold	Menyesuaikan dengan target kompresi, yaitu 30.6 %
Minimum Size	16
Output data	<p>Memulai kompresi ...</p> <p>Persentase kompresi saat ini: 89.376%</p> <p>Persentase kompresi saat ini: 89.376%</p> <p>Persentase kompresi saat ini: 60.992%</p> <p>Persentase kompresi saat ini: 28.1588%</p> <p>Persentase kompresi saat ini: 42.8824%</p> <p>Persentase kompresi saat ini: 35.251%</p> <p>Persentase kompresi saat ini: 31.3533%</p> <p>Persentase kompresi saat ini: 30.0615%</p> <p>Persentase kompresi saat ini: 30.7415%</p> <p>Persentase kompresi saat ini: 30.4809%</p>

	Persentase kompresi saat ini: 30.5775% Persentase kompresi saat ini: 30.6431% Persentase kompresi saat ini: 30.6638% Berhasil mencapai target dengan selisih 0.01! Membuat GIF kompresi... Ukuran file asli: 115889 bytes Ukuran file baru: 80420 bytes Waktu kompresi: 6940 ms Rasio kompresi: 30.606% Total node: 8033 Node daun: 6025 Kedalaman: 8
Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/chongqingVariance.gif
Test Case #7	
Input	
Metode	SSIM
Threshold	Menyesuaikan dengan target kompresi, yaitu 88.5 %

Minimum Size	16
Output data	<p>Memulai kompresi ...</p> <p>Persentase kompresi saat ini: 90.4456%</p> <p>Persentase kompresi saat ini: 89.0422%</p> <p>Persentase kompresi saat ini: 88.6938%</p> <p>Persentase kompresi saat ini: 88.6431%</p> <p>Persentase kompresi saat ini: 88.6862%</p> <p>Persentase kompresi saat ini: 88.73%</p> <p>Persentase kompresi saat ini: 88.7493%</p> <p>Threshold tidak dapat diturunkan lagi!!</p> <p>Berhasil mencapai target!</p> <p>Ukuran file asli: 2994357 bytes</p> <p>Ukuran file baru: 336863 bytes</p> <p>Waktu kompresi: 57286 ms</p> <p>Rasio kompresi: 88.7501%</p> <p>Total node: 84681</p> <p>Node daun: 63511</p> <p>Kedalaman: 9</p>
Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/spaceSSIM.gif
Test Case #8	

Input	
Metode	SSIM
Threshold	50
Minimum Size	100
Output data	Ukuran file asli: 497283 bytes Ukuran file baru: 232511 bytes Waktu kompresi: 14513 ms Rasio kompresi: 53.2437% Total node: 18417 Node daun: 13813 Kedalaman: 8
Output Image	
Output GIF	https://github.com/azfaradhi/Tucil2_13523115/blob/main/test/gif/lon donVariance.gif

BAB 5

ANALISIS

Berdasarkan pengujian terhadap beberapa metode tersebut, metode variance dan Mean Absolute Deviation (MAD) cenderung memberikan hasil kompresi yang lebih konservatif, dengan tetap mempertahankan banyak detail gambar. Sebaliknya, metode Max Pixel Difference menghasilkan rasio kompresi yang lebih tinggi, tetapi cenderung menghasilkan gambar yang relatif lebih kasar. Di lain sisi, penggunaan SSIM dapat memberikan keseimbangan efisiensi dan kualitas karena pada konsep dasarnya, metode ini mempertimbangkan persepsi manusia terhadap brightness, contrast, dan stucture.

Parameter threshold sangat berpengaruh terhadap jalannya kompresi. Semakin tinggi nilai sebuah threshold, semakin besar peluang suatu blok untuk dianggap seragam dan menghentikan proses rekursif. Di lain sisi, semakin rendah nilai sebuah threshold, semakin detail pengecekan keseragaman dari suatu blok, sehingga tetap mempertahankan detail dari gambar asli. Parameter minimum block size juga mempengaruhi batas akhir dari pembagian blok. Nilai yang terlalu besar dapat menghentikan proses rekursif lebih cepat dibandingkan nilai yang lebih kecil. Semakin kecil threshold dan minimum block size, maka akan semakin banyak jumlah node dan kadang dapat memperdalam pohon QuadTree.

Parameter lain yang mempengaruhi jalannya kompresi adalah ukuran dari gambar yang akan di proses. Gambar dengan banyak detail akan menghasilkan pohon yang lebih dalam dan lebih banyak bercabang, sementara citra dengan area homogen menghasilkan pohon yang lebih dangkal. Hal ini dapat dibuktikan dengan adanya variasi jumlah node total pada tiap percobaan.

Pada algoritma Divide and Conquer yang diimplementasikan (QuadTree::buildQuadTree), dibutuhkan $O(n^2)$ dalam menentukan nilai error dan $O(\log n)$ dalam melakukan rekursif. Sehingga, kompleksitas waktu dari algoritma ini adalah $O(n^2 \log n)$, dimana n adalah nilai maksimal dari tinggi atau lebar dari gambar.

BAB 6

IMPLEMENTASI BONUS

1. Implementasi persentase kompresi sebagai parameter tambahan

Teknik yang saya gunakan disini adalah binary search. Threshold awal ditetapkan pada pertengahan range masing-masing metode. Kemudian dihitung rasio setelah kompresinya. Jika rasio kurang dari target, maka pencarian akan bergeser ke kiri (yang lebih rendah) dan batas atas range berubah jadi nilai tengah sebelumnya. Jika rasio lebih besar dari target, maka pencarian akan bergeser ke kanan (yang lebih tinggi) dan batas bawah range berubah menjadi nilai tengah sebelumnya. Kasus khusus untuk metode SSIM, kedua kondisi sebelumnya dibalik.

```
while (true){  
    QuadTree quadtree(thersholdNow, minSize, method);  
    quadtree.startDNC(pixels);  
    vector<vector<RGB>> reconstructedImage = quadtree.reconstructImage(width,  
height);  
    img.saveImage(reconstructedImage, outputFileName);  
    newSize = img.getFileSize(outputFileName);  
  
    if (isTarget){  
        ratio = (double) (1 - (double)newSize/orginalSize)* 100;  
        if (abs(ratio - target) < 0.01){  
            cout << "Berhasil mencapai target dengan selisih 0.01!" << endl;  
            totalnodes = quadtree.totalNodes();  
            leafnodes = quadtree.leafNodes();  
            depth = quadtree.depth();  
            cout << "Membuat GIF kompresi..." << endl;  
            quadtree.createCompressionGIF(outputGifName , width, height);  
            break;  
        }  
        else if (abs(thersholdMax - thersholdMin) < 0.01){  
            cout << "Threshold tidak dapat diturunkan lagi! !" << endl;  
            cout << "Berhasil mencapai target!" << endl;  
            totalnodes = quadtree.totalNodes();  
            leafnodes = quadtree.leafNodes();  
            depth = quadtree.depth();  
            cout << "Membuat GIF kompresi..." << endl;  
            quadtree.createCompressionGIF(outputGifName , width, height);  
            break;  
        }  
  
        if (ratio < target){  
            if (method == 5){  
                thersholdMax = thersholdNow;  
                thersholdNow = (thersholdMax + thersholdMin) / 2;  
            }  
            else{  
                thersholdMin = thersholdNow;  
                thersholdNow = (thersholdMax + thersholdMin) / 2;  
            }  
        }  
        else if (ratio > target){  
            if (method == 5){  
                thersholdMin = thersholdNow;  
                thersholdNow = (thersholdMax + thersholdMin) / 2;  
            }  
            else{  
                thersholdMax = thersholdNow;  
                thersholdNow = (thersholdMax + thersholdMin) / 2;  
            }  
        }  
    }  
}
```

```

        }

        cout << "Percentase kompresi saat ini: " << ratio << "%" << endl;
        remove(outputFileName.c_str());
    }

    else{
        cout << "Berhasil mengkompresi gambar!" << endl;
        ratio = (double) (1 - (double)newSize/orginalSize)* 100;
        totalnodes = quadtree.totalNodes();
        leafnodes = quadtree.leafNodes();
        depth = quadtree.depth();
        cout << "Membuat GIF kompresi..." << endl;
        quadtree.createCompressionGIF(outputGifName, width, height);
        break;
    }
}

```

2. Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error
Penjelasan terkait pemilihan konstanta terdapat pada [Bab 1.2](#)

```

double QuadTree::errorSSIM(int x, int y, int height, int width) {

    RGB avg = calculateAverage(x, y, height, width);
    double uRx = avg.r;
    double uGx = avg.g;
    double uBx = avg.b;
    double uRy = avg.r;
    double uGy = avg.g;
    double uBy = avg.b;

    double sigmaRy = 0;
    double sigmaGy = 0;
    double sigmaBy = 0;
    double c1 = 6.5025;
    double c2 = 58.5225;

    double sigmaRx = 0;
    double sigmaGx = 0;
    double sigmaBx = 0;

    for (int i = y; i < y + height; i++){
        for (int j = x; j < x+width; j++){
            sigmaRx += (pixels[i][j].r - uRx) * (pixels[i][j].r - uRx);
            sigmaGx += (pixels[i][j].g - uGx) * (pixels[i][j].g - uGx);
            sigmaBx += (pixels[i][j].b - uBx) * (pixels[i][j].b - uBx);
        }
    }
    sigmaRx /= (height * width);
    sigmaGx /= (height * width);
    sigmaBx /= (height * width);

    double ssimr = ((2 * uRx * uRy + c1) * c2) / (((uRx * uRx) + (uRy * uRy) +
c1) * (sigmaRx + c2));
    double ssimg = ((2 * uGx * uGy + c1) * c2) / (((uGx * uGx) + (uGy * uGy) +
c1) * (sigmaGx + c2));
    double ssimb = ((2 * uBx * uBy + c1) * c2) / (((uBx * uBx) + (uBy * uBy) +
c1) * (sigmaBx + c2));

    return (ssimr + ssimg + ssimb) / 3;
}

```

3. Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar

Dengan menggunakan library gif.cpp, setiap tahapan dalam kompresi divisualisasikan menjadi satu frame dalam file GIF. Frame pertama dari GIF berisi gambar asli tanpa kompresi, yang digunakan sebagai referensi awal sebelum Quadtree dimulai. Kemudian QuadTree di traversal untuk mengunjungi semua node dengan BFS dan menyimpan semua node berdasarkan tingkatnya, yang kemudian digunakan untuk merekonstruksi gambar secara bertahap.

```
void QuadTree::createCompressionGIF(const std::string& outputPath, int width, int height) {

    int imgWidth = width;
    int imgHeight = height;

    GifWriter g;
    GifBegin(&g, outputPath.c_str(), imgWidth, imgHeight, 50);

    std::queue<std::shared_ptr<QuadTreeNode>> nodeQueue;
    std::vector<std::shared_ptr<QuadTreeNode>> allNodes;

    if (root) {
        nodeQueue.push(root);
    }

    std::vector<uint8_t> originalFrame(imgWidth * imgHeight * 4);
    for (int y = 0; y < imgHeight; y++) {
        for (int x = 0; x < imgWidth; x++) {
            int idx = (y * imgWidth + x) * 4;
            originalFrame[idx] = pixels[y][x].r;
            originalFrame[idx + 1] = pixels[y][x].g;
            originalFrame[idx + 2] = pixels[y][x].b;
        }
    }
    GifWriteFrame(&g, originalFrame.data(), imgWidth, imgHeight, 50);

    std::vector<std::vector<std::shared_ptr<QuadTreeNode>>> levelNodes;

    while (!nodeQueue.empty()) {
        int levelSize = nodeQueue.size();
        std::vector<std::shared_ptr<QuadTreeNode>> currentLevel;

        for (int i = 0; i < levelSize; i++) {
            auto node = nodeQueue.front();
            nodeQueue.pop();
            currentLevel.push_back(node);
            allNodes.push_back(node);

            if (!node->getIsLeaf()) {
                if (node->getTopLeft()) nodeQueue.push(node->getTopLeft());
                if (node->getTopRight()) nodeQueue.push(node->getTopRight());
                if (node->getBottomLeft()) nodeQueue.push(node->getBottomLeft());
                if (node->getBottomRight())
                    nodeQueue.push(node->getBottomRight());
            }
        }

        if (!currentLevel.empty()) {
            levelNodes.push_back(currentLevel);
        }
    }
}
```

```

}

for (size_t level = 0; level < levelNodes.size(); level++) {
    std::vector<std::vector<RGB>> reconstruction = pixels;
    for (size_t l = 0; l <= level; l++) {
        for (auto& node : levelNodes[l]) {
            if (node->getIsLeaf()) {
                for (int y = node->getY(); y < std::min(node->getY() +
node->getHeight(), imgHeight); y++) {
                    for (int x = node->getX(); x < std::min(node->getX() +
node->getWidth(), imgWidth); x++) {
                        reconstruction[y][x] = node->getAverageColor();
                    }
                }
            }
        }
    }

    std::vector<uint8_t> frame(imgWidth * imgHeight * 4);
    for (int y = 0; y < imgHeight; y++) {
        for (int x = 0; x < imgWidth; x++) {
            int idx = (y * imgWidth + x) * 4;
            frame[idx] = reconstruction[y][x].r;
            frame[idx + 1] = reconstruction[y][x].g;
            frame[idx + 2] = reconstruction[y][x].b;
            frame[idx + 3] = 255;
        }
    }

    GifWriteFrame(&g, frame.data(), imgWidth, imgHeight, 50);
}

std::vector<std::vector<RGB>> finalReconstruction =
reconstructImage(imgWidth, imgHeight);
std::vector<uint8_t> finalFrame(imgWidth * imgHeight * 4);
for (int y = 0; y < imgHeight; y++) {
    for (int x = 0; x < imgWidth; x++) {
        int idx = (y * imgWidth + x) * 4;
        finalFrame[idx] = finalReconstruction[y][x].r;
        finalFrame[idx + 1] = finalReconstruction[y][x].g;
        finalFrame[idx + 2] = finalReconstruction[y][x].b;
        finalFrame[idx + 3] = 255;
    }
}

GifWriteFrame(&g, finalFrame.data(), imgWidth, imgHeight, 50);

GifEnd(&g);
}

```

LAMPIRAN

Tautan Repository: https://github.com/azfaradhi/Tucil2_13523115

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 (kelompok) sendiri	✓	

Referensi:

- https://en.cppreference.com/w/cpp/memory/shared_ptr
- <https://ieeexplore.ieee.org/document/1284395>
- https://www.itu.int/dms_pubrec/itu-r/rec/bt/r-rec-bt.601-7-201103-i!!pdf-e.pdf