

LAPORAN TUGAS KECIL 2
IF2211 STRATEGI ALGORITMA
Kompresi Gambar Dengan Metode Quadtree



Disusun Oleh:

Aryo Bama Wiratama (13523088)

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

A. Algoritma Divide and Conquer

Algoritma divide and conquer adalah algoritma yang bekerja dengan cara membagi suatu persoalan menjadi persoalan yang lebih kecil dari persoalan semula, menyelesaikan masing-masing upa-persoalan, kemudian menggabungkan upa-persoalan untuk mendapat solusi dari persoalan awal. Algoritma ini memiliki tiga langkah utama, yaitu:

1. Divide

Membagi persoalan menjadi beberapa upa-persoalan yang lebih kecil dan mirip dengan persoalan awal

2. Conquer

Menyelesaikan masing-masing upa-persoalan secara langsung. Jika upa-persoalan masih cukup besar divide lagi secara rekursif.

3. Combine

Menggabungkan solusi semua upa-persoalan sehingga membentuk solusi dari persoalan awal

B. Quadtree

Quadtree merupakan struktur data berjenjang yang berfungsi untuk memecah ruang atau data menjadi segmen-segmen lebih kecil, dan sering diaplikasikan dalam pemrosesan citra. Ketika digunakan untuk kompresi gambar, Quadtree membagi citra menjadi blok-blok kecil berdasarkan homogenitas warna atau nilai intensitas piksel. Proses ini dimulai dengan membagi gambar menjadi empat bagian, kemudian mengevaluasi apakah setiap kuadran memiliki keseragaman nilai berdasarkan analisis komponen warna RGB, dengan cara membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel dalam kuadran tersebut. Apabila suatu bagian tidak memiliki nilai yang homogen, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman yang diinginkan atau hingga mencapai ukuran blok minimum yang telah ditetapkan.

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.

Algoritma divide and conquer dapat digunakan untuk mengimplementasikan kompresi menggunakan quadtree. Berikut adalah ide penggunaan algoritma divide conquer dalam pengkompresian gambar menggunakan struktur data quadtree:

1. **Solve:**

Jika blocksize (ukuran block) \leq minblock ATAU suatu blok dianggap homogen (umumnya $\text{error} < \text{threshold}$), lakukan normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

2. **Divide:**

Jika blocksize (ukuran block) \geq minblock DAN suatu blok dianggap tidak homogen (umumnya $\text{error} > \text{threshold}$), bagi blok menjadi 4 subblok sama besar.

3. **Conquer**

Untuk setiap subblok, ulangi proses **solve** dan **divide** secara rekursif.

4. **Combine**

Setelah semua subblok diproses, gabungkan informasi dari subblok-subblok pada struktur quadtree untuk membentuk suatu gambar yang telah terkompresi.

Berikut pseudocode:

```

function Construct(block, x, y, width, height):
    error ← CountError(block)

    if errorMethod ≠ 5:
        isHomogeneous ← (error < threshold)
    else:
        isHomogeneous ← (error > threshold)

    if isHomogeneous OR (width × height / 4 < minBlock):
        color ← NormalizeColor(block)
        return NewLeafNode(color, x, y, width, height)

    w1 ← width // 2
    w2 ← width - w1
    h1 ← height // 2
    h2 ← height - h1

    topLeft ← Construct(CopyBlock(block, 0, 0, w1, h1), x, y, w1, h1)
    topRight ← Construct(CopyBlock(block, w1, 0, w2, h1), x + w1, y, w2, h1)
    bottomLeft ← Construct(CopyBlock(block, 0, h1, w1, h2), x, y + h1, w1, h2)
    bottomRight ← Construct(CopyBlock(block, w1, h1, w2, h2), x + w1, y + h1, w2, h2)

    return NewInternalNode(topLeft, topRight, bottomLeft, bottomRight, x, y, width, height)

```

```

function MergeBlock(node, result, width, height):
    if width ≠ result[0].length OR height ≠ result.length:
        print("Height or width doesn't match with vector")
        return

    if node is Leaf:
        for i from 0 to node.height - 1:
            for j from 0 to node.width - 1:
                result[node.y + i][node.x + j] ← node.color
    else:
        MergeBlock(node.topLeft, result, width, height)
        MergeBlock(node.topRight, result, width, height)
        MergeBlock(node.bottomLeft, result, width, height)
        MergeBlock(node.bottomRight, result, width, height)

```

C. Source Code

Projek ini ditulis menggunakan bahasa c++. Projek ini menggunakan single header library untuk memudahkan dalam melakukan image processing, sebagai berikut:

1. stb_image.h

Digunakan untuk membaca gambar

Berikut source code dari library di atas:

https://github.com/nothings/stb/blob/master/stb_image.h

2. stb_write_image.h

Digunakan untuk membuat gambar

Berikut source code dari library di atas:

https://github.com/nothings/stb/blob/master/stb_image_write.h

3. gif.h

Digunakan untuk membuat gif

Berikut source code dari library di atas:

<https://github.com/charlietangora/gif-h/blob/master/gif.h#L766>

Tentu saja penggunaan library dimaksudkan untuk mempermudah proses dalam tugas ini. Penggunaan library ini bukan untuk mencari hasil instan, melainkan sebagai **alat bantu yang efisien** agar dapat lebih fokus pada logika inti dari program yang saya kerjakan.

Projek ini dibagi menjadi beberapa file, yaitu:

1. Color.hpp dan Color.cpp

Berisi implementasi class Color yang digunakan sebagai **representasi warna dalam format RGB**, di mana setiap komponen warna (merah, hijau, biru) disimpan sebagai nilai numerik. Class Color juga bisa menyimpan informasi alpha suatu gambar.

2. QuadtreeNode.hpp dan QuadtreeNode.cpp

Berisi implementasi dari class QuadtreeNode. Class ini digunakan sebagai **struktur dasar dari pohon quadtree**, yang merepresentasikan sebuah simpul (node) dalam hierarki pembagian ruang atau gambar.

3. Quadtree.hpp dan Quadtree.cpp

Berisi implementasi dari class Quadtree. Class ini digunakan sebagai **pengelola struktur data quadtree secara keseluruhan**, yang membentuk dan menyusun pohon dari node-node (QuadtreeNode) berdasarkan pembagian citra atau ruang ke dalam blok-blok yang lebih kecil.

4. Utility.hpp dan Utility.cpp

Berisi implementasi dari semua fungsi penting yang dibutuhkan dalam program.

5. Main.cpp

Berisi **fungsi main() sebagai titik awal eksekusi program**, yang bertanggung jawab untuk mengatur alur kerja utama dari program kompresi gambar. File ini akan meminta input dari pengguna serta mengeluarkan output yang diharapkan.

Berikut source codenya

1. Color.hpp dan Color.cpp

```
#ifndef COLOR_HPP
#define COLOR_HPP

#include <iostream>
using namespace std;

class Color{
private:
    int r;
    int g;
    int b;
    int a;
    int channels;
public:
    Color();

    Color(int r, int g, int b);

    Color(int r, int g, int b, int a);

    Color(const Color& other);

    ~Color();

    Color& operator=(const Color& other);

    int getRed() const;

    int getBlue() const;

    int getGreen() const;

    int getAlpha() const;

    int getChannels() const;
};
```

```

        void setRed(int r);

        void setBlue(int b);

        void setGreen(int g);

        void setAlpha(int a);
    };

#endif

```

```

#include "Color.hpp"

Color::Color() : r(0), g(0), b(0), channels(0){
}

Color::Color(int r, int g, int b) : r(r), g(g), b(b), channels(3){
}

Color::Color(int r, int g, int b, int a): r(r), g(g), b(b), a(a), channels(4){
}

Color::Color(const Color& other){
    r = other.r;
    g = other.g;
    b = other.b;
    channels = other.channels;

    if(channels == 4) a = other.a;
}

Color::~Color(){
}

Color& Color::operator=(const Color& other){
    r = other.r;
    g = other.g;
    b = other.b;
    channels = other.channels;

    if(channels == 4) a = other.a;
    return *this;
}

```

```

int Color::getRed() const{
    return r;
}

int Color::getBlue() const{
    return b;
}

int Color::getGreen() const{
    return g;
}

int Color::getAlpha() const{
    return a;
}

int Color::getChannels() const{
    return channels;
}

void Color::setRed(int r) {
    this->r = r;
}

void Color::setBlue(int b){
    this->b = b;
}

void Color::setGreen(int g){
    this->g = g;
}

void Color::setAlpha(int a){
    this->a = a;
}

```

2. QuadtreeNode.hpp dan QuadtreeNode.cpp


```

#ifndef QUADREENODE_HPP
#define QUADREENODE_HPP

#include <iostream>
#include "Color.hpp"

using namespace std;

class QuadreeNode{
private:
    Color color;
    int x;
    int y;
    int width;
    int height;

    QuadreeNode* topLeft;
    QuadreeNode* topRight;
    QuadreeNode* bottomLeft;
    QuadreeNode* bottomRight;

public:
    bool isLeaf;

    QuadreeNode(Color color, int x, int y, int width, int height);

    QuadreeNode(QuadreeNode *tl, QuadreeNode *tr, QuadreeNode *bl, QuadreeNode *br, int x, int y, int width, int height);

    ~QuadreeNode();

    QuadreeNode* getTopLeft();

    QuadreeNode* getTopRight();

    QuadreeNode* getBottomLeft();

    QuadreeNode* getBottomRight();

    int getWidth();

    int getHeight();

    int getX();

    int getY();

    Color getColor();
};

#endif

```

3. Quadtree.hpp dan Quadtree.cpp

```

#ifndef QUADTREE_HPP

class Quadtree{

private:
    int errorMethod;
    double threshold;
    int minBlock;
    int percentage;
    QuadtreeNode *root;

    vector<vector<Color>> copyBlock(const vector<vector<Color>> &block, int x, int y, int width, int height);

    double countError(const vector<vector<Color>> & image);

    Color normalizeColor(const vector<vector<Color>> & block);

    QuadtreeNode* construct(const vector<vector<Color>> & block, int x, int y, int width, int height);

public:

    void countVariance(const vector<vector<Color>> & image, double &varianceRed, double &varianceGreen, double &varianceBlue);

    Quadtree(const vector<vector<Color>> & image, int errorMethod, double threshold, int minBlock, int percentage);

    QuadtreeNode* getRoot();

    int countDepth(QuadtreeNode *node, int cnt);

    int countNode(QuadtreeNode *node);

    int countLeaf(QuadtreeNode *node);

};

#endif

```

```

#include "Quadtree.hpp"
#include <math.h>

vector<vector<Color>> Quadtree:: copyBlock(const vector<vector<Color>> &block, int x, int y, int width, int height){

    vector<vector<Color>> result(height, vector<Color>(width));

    for (int i = 0; i < height; i++){
        for (int j = 0; j < width; j++){
            result[i][j] = block[y + i][x + j];
        }
    }
    return result;
}

void Quadtree::countVariance(const vector<vector<Color>> & image, double &varianceRed, double &varianceGreen, double &varianceBlue){

    double meanRed = 0;
    double meanGreen = 0;
    double meanBlue = 0;
    varianceRed = varianceGreen = varianceBlue = 0;

    double size = image.size() * image[0].size();

    for (size_t i = 0; i < image.size(); i++){
        for (size_t j = 0; j < image[0].size(); j++){
            meanRed += image[i][j].getRed();
            meanGreen += image[i][j].getGreen();
            meanBlue += image[i][j].getBlue();
        }
    }

    meanRed /= size;
    meanGreen /= size;
    meanBlue /= size;
}

```

4. utility.hpp dan utility.cpp

```

#ifndef UTILITY_HPP
#define UTILITY_HPP

#include "library/stb_image.h"
#include <vector>
#include "Color.hpp"
#include "QuadTree.hpp"

vector<vector<Color>> imageToVector (unsigned char* data, int width, int height, int channels);

void mergeBlock(QuadtreeNode* node, vector<vector<Color>>&result, int width, int height);

unsigned char* convertToByteArray(const vector<vector<Color>>& image, int& width, int& height, int& channels);

void image_write_func(void* context, void* data, int size);

unsigned char* imageByCompressionRate(const vector<vector<Color>>& image, double compressionRate, int errorMethod, uintmax_t sizeBytes,
    int &nNode, int &depthTree, int channels);

void createGif(const vector<std::vector<Color>> &image, unsigned char* compressImageData, const char* filename, int width, int height, int depthTree, int channels);

string getFileExtension(const std::string& filePath);

#endif

```

```

#include "utility.hpp"
#include "library/stb_image_write.h"
#include "library/gif.h"

vector<vector<Color>> imageToVector (unsigned char* data, int width, int height, int channels){
    if (data == nullptr){
        cout << "Failed to convert image" << endl;
        return {};
    }

    vector<vector<Color>> result(height, vector<Color>(width));

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = (y * width + x) * channels;
            result[y][x].setRed(data[index]);
            result[y][x].setGreen(data[index+1]);
            result[y][x].setBlue(data[index+2]);
            if (channels == 4) result[y][x].setAlpha(data[index+3]);
        }
    }
    return result;
}

void mergeBlock(QuadtreeNode* node, vector<vector<Color>>&result, int width, int height){

    if (width != (int)result[0].size() || height != (int) result.size()){
        cout << "Height or width doesn't match with vector" << endl;
        return;
    }

    if(node->isLeaf){
        for (int i = 0; i < node->getHeight(); i++){
            for (int j = 0; j < node->getWidth(); j++){
                result[node->getY() + i][node->getX() + j] = node->getColor();
            }
        }
    }
}

```

```

        for (int j = 0; j < node->getWidth(); j++){
            result[node->getY() + i][node->getX() + j] = node->getColor();
        }
    }else{
        mergeBlock(node->getTopLeft(), result, width, height);
        mergeBlock(node->getTopRight(), result, width, height);
        mergeBlock(node->getBottomLeft(), result, width, height);
        mergeBlock(node->getBottomRight(), result, width, height);
    }
}

}

unsigned char* convertToByteArray(const vector<vector<Color>>& image, int& width, int& height, int& channels) {
    width = image[0].size();
    height = image.size();

    unsigned char* imageData = new unsigned char[width * height * channels];

    int idx = 0;
    for (int i = 0; i < height; i++) {
        for (int j = 0; j < width; j++) {
            imageData[idx++] = static_cast<unsigned char>(std::min(255, std::max(0, image[i][j].getRed())));
            imageData[idx++] = static_cast<unsigned char>(std::min(255, std::max(0, image[i][j].getGreen())));
            imageData[idx++] = static_cast<unsigned char>(std::min(255, std::max(0, image[i][j].getBlue())));
            if (channels == 4) idx++;
        }
    }

    return imageData;
}

```

```

void image_write_func(void* context, void* data, int size) {
    vector<unsigned char*> buf = reinterpret_cast<vector<unsigned char*>>(context);
    unsigned char* byteData = reinterpret_cast<unsigned char*>(data);
    buf->insert(buf->end(), byteData, byteData + size);
}

unsigned char* imageByCompressionRate(const vector<vector<Color>>& image, double compressionRate, int errorMethod, uintmax_t sizeBytes,
int &nNode, int &depthTree, int channels){

    int maxThreshold = 0, minThreshold = 0, maxRange = 0, maxActualThreshold = 0;
    double tolerance = 0;
    compressionRate *= 100;
    double percentage = 0;

    unsigned char* imageData = nullptr;

    int height = image.size(), width = image[0].size();

    if (errorMethod == 1){
        maxRange = maxThreshold = 1000;
        tolerance = 5;
        maxActualThreshold = 1000;
    }else if (errorMethod == 2){
        maxRange = maxThreshold = 10000;
        tolerance = 5;
        maxActualThreshold = 127;
    }else if (errorMethod == 3){
        maxRange = maxThreshold = 10000;
        tolerance = 5;
        maxActualThreshold = 255;
    }else if (errorMethod == 4){
        maxRange = maxThreshold = 100000;
        tolerance = 5;
        maxActualThreshold = 8;
    }else if (errorMethod == 5){
        maxRange = maxThreshold = 1000;
    }
}

```

```

do{

    if (imageData != nullptr) {
        delete[] imageData;
    }

    int mid = (maxThreshold + minThreshold)/2;

    double actualThreshold = ((double)mid / maxRange) * maxActualThreshold;

    Quadtree quadtree(image,errorMethod,actualThreshold,2,compressionRate/100);

    vector<vector<Color>> compression(height, vector<Color>(width));

    mergeBlock(quadtree.getRoot(), compression, width, height);

    imageData = convertToByteArray(compression, width, height, channels);

    vector<unsigned char> imageBuffer;
    stbi_write_jpg_to_func(image_write_func, &imageBuffer, width, height, 3, imageData, 90);

    percentage = ((double)imageBuffer.size()/sizeBytes) * 100;

    if (percentage > compressionRate){
        if (errorMethod != 5){
            minThreshold = mid; // kita naikkan thresholdnya biar kualitas makin jelek
        }else{
            maxThreshold = mid;
        }
    }else{
        if (errorMethod != 5){
            maxThreshold = mid; // kita turunkan thresholdnya biar kualitas makin bagus
        }else{
            minThreshold = mid;
        }
    }
}

```

```

// cout << "Percentage: " << percentage << endl;
// cout << "abs: " << abs(percentage - compressionRate) << endl;

nNode = quadtree.countNode(quadtree.getRoot());
depthTree = quadtree.countDepth(quadtree.getRoot(),0);

int loading = percentage/compressionRate * 100;

cout << "Loading... " << loading << "%\n";

}while(minThreshold < maxThreshold && abs(percentage - compressionRate) > tolerance);

return imageData;
}

void createGif(const vector<std::vector<Color>> &image, unsigned char* compressImageData, const char* filename, int width, int height, int depthTree, int channels) {
    GifWriter writer;
    GifBegin(&writer, filename, width, height, 10);
    int minBlock = width * height;

    int loading = 0;

    cout << "Membuat GIF...\n";
    cout << "Loading... " << loading << "%\n";

    uint8_t trans_r = 0, trans_g = 0, trans_b = 0;

    for (int i = 0; i < max(2, depthTree - 2); i++) {
        loadline = (i + 1) * 10;

```

```

for (int i = 0; i < max(2, depthtree - 2); i++) {
    loading = (i + 1) * 10;
    cout << "Loading... " << loading << " %\n";

    vector<vector<Color>> frame(height, vector<Color>(width));
    vector<uint8_t> frameData(width * height * 4);

    Quadtree quadtree(image, 6, 0, minBlock, 0);
    mergeBlock(quadtree.getRoot(), frame, width, height);

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            const Color& c = frame[y][x];
            int index = (y * width + x) * 4;

            uint8_t alpha = (channels == 4) ? image[y][x].getAlpha() : 255;

            if (alpha < 128) {
                frameData[index + 0] = trans_r;
                frameData[index + 1] = trans_g;
                frameData[index + 2] = trans_b;
            } else {
                frameData[index + 0] = c.getRed();
                frameData[index + 1] = c.getGreen();
                frameData[index + 2] = c.getBlue();
            }

            frameData[index + 3] = 255;
        }
    }

    minBlock /= 4;
    GifWriteFrame(&writer, frameData.data(), width, height, 100);
}

vector<uint8_t> frameData(width * height * 4);

```

```

vector<uint8_t> frameData(width * height * 4);
for (int i = 0, j = 0; i < width * height * channels; i += channels, j += 4) {
    uint8_t r = compressImageData[i + 0];
    uint8_t g = compressImageData[i + 1];
    uint8_t b = compressImageData[i + 2];
    uint8_t a = (channels == 4) ? compressImageData[i + 3] : 255;

    if (a < 128) {
        frameData[j + 0] = trans_r;
        frameData[j + 1] = trans_g;
        frameData[j + 2] = trans_b;
    } else {
        frameData[j + 0] = r;
        frameData[j + 1] = g;
        frameData[j + 2] = b;
    }

    frameData[j + 3] = 255;
}

cout << "Complete... " << 100 << " %\n";
GifWriteFrame(&writer, frameData.data(), width, height, 100);

GifEnd(&writer);
}

string getFileExtension(const std::string& filePath) {
    size_t dotPos = filePath.find_last_of(".");
    if (dotPos != std::string::npos && dotPos != filePath.length() - 1) {
        return filePath.substr(dotPos + 1);
    }
    return "";
}

```

5. main.cpp

```
#include "Color.hpp"
#include "QuadtreeNode.hpp"
#include "Quadtree.hpp"
#include "utility.hpp"
#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "library/stb_image.h"
#include "library/stb_image_write.h"
#include <chrono>
#include <filesystem>
using namespace std::chrono;

int main(){

    string addressPath, savePath, GIFPath, extension = "";
    int errorMethod = 0;
    double compressionRate = 0;
    unsigned char* imageData;
    double percentage = 0;
    int nNode = 0, depthTree = 0;

    double executeTime = 0;

    int width, height, channels;

    cout << "Masukkan alamat gambar yang ingin dikompresi: ";
    getline(cin, addressPath);

    extension = getFileExtension(addressPath);

    cout << "Masukkan alamat tempat menyimpan gambar: ";
    getline(cin, savePath);

    cout << "Masukkan alamat tempat menyimpan GIF: ";
    getline(cin, GIFPath);

    unsigned char* data = stbi_load(addressPath.c_str(), &width, &height, &channels, 0);
```

```

vector<vector<Color>> image = imageToVector(data,width,height,channels);

uintmax_t sizeBytes = filesystem::file_size(addressPath);

do{

    cout << "Berikut error method yang bisa kamu pilih: \n";
    cout << "1. Variance\n";
    cout << "2. Median Absolute Deviation\n";
    cout << "3. Max Pixel Difference\n";
    cout << "4. Enthrophy\n";
    cout << "5. SSIM\n";
    cout << "Pilih error method : ";
    cin >> errorMethod;

    if (cin.fail()) {
        cout << "Itu bukan angka, coba lagi!\n";
        cin.clear();
        cin.ignore(1000, '\n');
    }else if (errorMethod < 1 && errorMethod > 5){
        cout << "Pilih angka di rentang 1 - 5\n";
    }

}while (errorMethod < 1 && errorMethod > 5);

while (true) {
    cout << "Masukkan compression rate: ";
    cin >> compressionRate;

    if (cin.fail()) {
        cout << "Itu bukan angka, coba lagi!\n";
        cin.clear();
        cin.ignore(1000, '\n');
    } else {

```

```

if (compressionRate == 0){

    double threshold = 0;
    int minBlock = 1;

    while (true) {
        cout << "Masukkan threshold: ";
        cin >> threshold;
        if (cin.fail()) {
            cout << "Itu bukan angka, coba lagi!\n";
            cin.clear();
            cin.ignore(1000, '\n');
        } else {
            break;
        }
    }

    while (true) {
        cout << "Masukkan minimal block size: ";
        cin >> minBlock;

        if (cin.fail()) {
            cout << "Itu bukan angka, coba lagi!\n";
            cin.clear();
            cin.ignore(1000, '\n');
        } else {
            break;
        }
    }

    cout << "Sedang mengkompresi...\n";
    cout << "Loading... 0%\n";

```



```

        auto start = high_resolution_clock::now();
        Quadtree quadtree(image,errorMethod,threshold,minBlock,0);

        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<milliseconds>(stop - start);
        executeTime = duration.count();

        vector<vector<Color>> compression(height, vector<Color>(width));

        mergeBlock(quadtree.getRoot(), compression, width, height);

        nNode = quadtree.countNode(quadtree.getRoot());
        depthTree = quadtree.countDepth(quadtree.getRoot(),0);

        imageData = convertToByteArray(compression, width, height, channels);
    }else{

        auto start = high_resolution_clock::now();
        cout << "Sedang mengkompresi...\n";
        cout << "Loading... 0%\n";
        imageData = imageByCompressionRate(image, compressionRate, errorMethod, sizeBytes, nNode, depthTree, channels);
        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<milliseconds>(stop - start);
        executeTime = duration.count();
    }

    cout << "Complete... 100%\n";

    stbi_image_free(data);

```

```

    if (extension == "png"){

        if (channels == 4) {
            for (int y = 0; y < height; y++) {
                for (int x = 0; x < width; x++) {
                    int index = (y * width + x) * 4;
                    imageData[index + 3] = image[y][x].getAlpha();
                }
            }

            stbi_write_png(savePath.c_str(), width, height, channels, imageData, width * channels);
        }else{
            stbi_write_jpg(savePath.c_str(), width, height, channels, imageData, 90);
        }

        auto start = high_resolution_clock::now();

        createGif(image,imageData,GIFPath.c_str(),width, height, depthTree, channels);

        auto stop = high_resolution_clock::now();
        auto duration = duration_cast<milliseconds>(stop - start);
        executeTime += duration.count();

        cout << "Waktu eksekusi: " << executeTime << " ms" << endl;

        cout << "Ukuran gambar sebelum: " << sizeBytes << " Bytes" << endl;
        cout << "Ukuran gambar sesudah: " << filesystem::file_size(savePath) << " Bytes" << endl;

        percentage = (1 -((double) filesystem::file_size(savePath) / sizeBytes)) * 100;

        cout << "Persentase kompresi: " << percentage << " %" << endl;
    }

```

```

cout << "Node: " << nNode << endl;
cout << "depth: " << depthTree << endl;

delete[] imageData;
}

```

D. Testing

1. Test 1

Input:

```

Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input1.jpg
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output1.jpg
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output1.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Entrophy
5. SSIM
Pilih error method : 1
Masukkan compression rate: 0
Masukkan threshold: 50
Masukkan minimal block size: 8

```



Output:

```

Sedang mengkompresi...
Loading... 0%
Loading... 0%
Complete... 100%
Complete... 100%
Membuat GIF...
Membuat GIF...
Loading... 0 %
Loading... 10 %
Loading... 20 %
Loading... 30 %
Loading... 40 %
Loading... 50 %
Loading... 60 %
Loading... 70 %
Complete... 100 %
Waktu eksekusi: 18847 ms
Ukuran gambar sebelum: 491026 Bytes
Ukuran gambar sesudah: 284220 Bytes
Persentase kompresi: 42.1171 %
Node: 71384
depth: 8

```



2. Test 2

Input:

```

Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input1.jpg
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output2.jpg
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output2.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Enthrophy
5. SSIM
Pilih error method : 1
Masukkan compression rate: 0.75

```



Output:

```
Sedang mengkompresi...
Loading... 0%
Loading... 17%
Loading... 17%
Loading... 17%
Loading... 18%
Loading... 21%
Loading... 32%
Loading... 49%
Loading... 63%
Loading... 73%
Loading... 82%
Loading... 88%
Loading... 93%
Loading... 97%
Complete... 100%
Membuat GIF...
Loading... 0 %
Loading... 10 %
Loading... 20 %
Loading... 30 %
Loading... 40 %
Loading... 50 %
Loading... 60 %
Loading... 70 %
Loading... 80 %
Complete... 100 %
Waktu eksekusi: 106651 ms
Ukuran gambar sebelum: 491026 Bytes
Ukuran gambar sesudah: 359639 Bytes
Persentase kompresi: 26.7576 %
Node: 293248
depth: 9
```



3. Test 3

Input:

```
Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input3.png
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output3.png
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output3.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Enthrophy
5. SSIM
Pilih error method : 2
Masukkan compression rate: 0
Masukkan threshold: 15
Masukkan minimal block size: 4
Cedeng-masah-masah
```



Output:

```
Sedang mengkompresi...  
Loading... 0%  
Complete... 100%  
Membuat GIF...  
Loading... 0 %  
Loading... 10 %  
Loading... 20 %  
Loading... 30 %  
Loading... 40 %  
Loading... 50 %  
Loading... 60 %  
Loading... 70 %  
Loading... 80 %  
Complete... 100 %  
Waktu eksekusi: 48246 ms  
Ukuran gambar sebelum: 850756 Bytes  
Ukuran gambar sesudah: 377373 Bytes  
Persentase kompresi: 55.6426 %  
Node: 30624  
depth: 9
```



4. Test 4

Input:

```
Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input3.png
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output4.png
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output4.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Enthrophy
5. SSIM
Pilih error method : 3
Masukkan compression rate: 0
Masukkan threshold: 40
Masukkan minimal block size: 10
```



Output:


```
Waktu eksekusi: 34801 ms
Ukuran gambar sebelum: 850756 Bytes
Ukuran gambar sesudah: 495097 Bytes
Persentase kompresi: 41.8051 %
Node: 72296
depth: 9
```



5. Test 5

Input:

```
Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input4.png
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output5.png
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output5.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Enthrophy
5. SSIM
Pilih error method : 3
Masukkan compression rate: 0
Masukkan threshold: 4
Masukkan minimal block size: 8
```



Output:

```
Waktu eksekusi: 7802 ms
Ukuran gambar sebelum: 1699494 Bytes
Ukuran gambar sesudah: 486360 Bytes
Persentase kompresi: 71.3821 %
Node: 158736
depth: 8
```



6. Test 6

Input:

```
Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input4.png
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output6.png
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output6.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Entrophy
5. SSIM
Pilih error method : 4
Masukkan compression rate: 0
Masukkan threshold: 2
Masukkan minimal block size: 8
```



Output:

```
Waktu eksekusi: 8292 ms
Ukuran gambar sebelum: 1699494 Bytes
Ukuran gambar sesudah: 456879 Bytes
Persentase kompresi: 73.1168 %
Node: 139156
depth: 8
```



7. Test 7

Input:

```
Masukkan alamat gambar yang ingin dikompresi: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/input5.png
Masukkan alamat tempat menyimpan gambar: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output7.png
Masukkan alamat tempat menyimpan GIF: C:/Users/AryoBama/Jurusan/Semester_4/Stima/Tucil/Tucil2_13523088/test/output7.gif
Berikut error method yang bisa kamu pilih:
1. Variance
2. Median Absolute Deviation
3. Max Pixel Difference
4. Enthrophy
5. SSIM
Pilih error method : 5
Masukkan compression rate: 0
Masukkan threshold: 0.3
Masukkan minimal block size: 6
```



Output:

```
Waktu eksekusi: 1606 ms
Ukuran gambar sebelum: 544015 Bytes
Ukuran gambar sesudah: 91351 Bytes
Persentase kompresi: 83.208 %
Node: 21828
depth: 7
```



E. Analisis

1. Kompleksitas waktu

Misal terdapat gambar berukuran $W \times H$ dengan ukuran minimal block $A \times A$. Untuk kasus terburuk (tidak ada blok yang homogen) setiap blok akan dibagi hingga mencapai ukuran A^2 .

Kedalaman rekursi maksimumnya adalah:

Blok awal $W \times H$

Setelah 1 pembagian: $W/2 \times H/2$

Setelah 2 pembagian: $W/4 \times H/4$

.....

Setelah k pembagian: $W/2^k \times H/2^k$

Pembagian berhenti saat $W/2^k \leq a$ dan $H/2^k \leq a$.

Misal $W \geq H$ maka:

$$W/2^k \leq a \rightarrow k \geq \log_2(W/a)$$

Jadi kedalaman rekursi maksimum adalah $O(\log(W/a))$.

Pada setiap level k dari rekursi:

Jumlah node maksimum: 4^k

Operasi per node: $O(1)$ untuk pemeriksaan homogenitas

Jumlah total operasi: $\sum(4^k)$ dari $k=0$ sampai $\log_2(W/a)$

Sigma diatas adalah deret geometri dengan rasio 4 sehingga total operasi adalah $(4^{(\log_2(W/a)+1)} - 1)/3$

Sederhanakan ekspresi di atas sehingga total menjadi $(W/a)^2$

Namun, karena pembagian blok harus mempertimbangkan W dan juga H bukan $\max(W,H)$ maka total operasi adalah $(W \times H) / a^2$

Sehingga kompleksitas algoritmanya adalah $O((W \times H) / a^2)$

2. Kompleksitas ruang

Banyaknya node akan sama dengan banyak operasi, sehingga total node adalah $(W \times H) / a^2$ Misal tiap node memiliki kompleksitas ruang $O(1)$, maka kompleksitas ruang algoritma ini adalah $O((W \times H) / a^2)$.

F. Bonus

1. SSIM

SSIM (Structural Similarity Index Measure) adalah **metrik** yang digunakan untuk mengukur **kemiripan visual antara dua gambar**.

SSIM mempertimbangkan beberapa hal, yaitu:

- Luminance (kecerahan)
- Contrast (kontras)
- Structure (tekstur)

Nilai SSIM bisa berada pada rentang 0 hingga 1. Semakin mendekati 1 artinya suatu gambar semakin mirip, begitupun sebaliknya. Namun untuk kasus yang sangat ekstrem nilai SSIM bisa berada di sekitar -1. Hal itu dapat terjadi apabila 2 buah gambar sangat berlawanan, misalnya suatu gambar sangat cerah, tapi gambar yang lainnya sangat gelap.

SSIM dapat dihitung dari rumus berikut:

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(2\sigma_{xy} + C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_x^2 + \sigma_y^2 + C_2)}$$

Namun karena pada program ini blok yang dibandingkan adalah blok sebelum dinormalisasi dan sesudah dinormalisasi (blok yang homogen) kita dapat menyimpulkan bahwa nilai rata – rata blok homogen sama dengan nilai tiap pixelnya. Oleh karena itu varian untuk blok homogen adalah 0. Oleh karena itu nilai kovarian juga 0. Dengan demikian, rumus dapat disederhanakan menjadi

$$SSIM(x, y) = \frac{(2\mu_x\mu_y + C_1)(C_2)}{(\mu_x^2 + \mu_y^2 + C_1)(\sigma_y^2 + C_2)}$$

Perhatikan bahwa nilai rata – rata blok sebelum di normalisasi akan sama dengan rata – rata blok sesudah di normalisasi karena normalisasi dihitung berdasarkan rata – rata blok itu sendiri. Dengan demikian rata – rata sebelum akan sama dengan rata – rata sesudah. Rumus dapat disederhanakan

$$\begin{aligned}\mu_x &= \mu_y \\ SSIM(x, y) &= \frac{(2\mu_y^2 + C_1)(C_2)}{(\mu_y^2 + \mu_y^2 + C_1)(\sigma_y^2 + C_2)} \\ SSIM(x, y) &= \frac{(2\mu_y^2 + C_1)(C_2)}{(2\mu_y^2 + C_1)(\sigma_y^2 + C_2)} \\ SSIM(x, y) &= \frac{(C_2)}{(\sigma_y^2 + C_2)}\end{aligned}$$

Nilai C_2 adalah sebuah konstanta yang digunakan agar ketika kovarian mendekati 0 nilai SSIM tidak terlalu besar. Nilai C_2 dapat dihitung dari $(K_2 \times L)^2$. Nilai K_2 yang digunakan umumnya adalah 0.03 nilai ini didapatkan dari hasil eksperimen. Nilai L adalah dynamic range yang dalam hal ini adalah 255 (24 bit RGB dan 8 bit per kanal).

Nilai SSIM total dapat dihitung sebagai berikut

$$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$$

Di Mana nilai masing berturut-turut adalah 0.2989, 0.5870, 0.1140. Nilai ini mengikuti luminance perception.

2. Compression rate

Untuk mendapatkan nilai threshold berdasarkan pesentase kompresi yang diinput pengguna, kita harus menelusuri semua threshold yang mungkin untuk tiap metode. Cara itu tentunya akan memakan waktu yang lama oleh karena itu ide yang akan digunakan adalah menggunakan metode binary search.

Diinisialisasi $\text{minThreshold} = 0$ dan $\text{maxThreshold} = x$ (maxThreshold berbeda tiap metode). Ambil angka tengah antara minThreshold dan maxThreshold . Kemudian kompresi gambar menggunakan quadtree. Hitung persentase kompresinya jika hasil gambar belum memenuhi compression rate yang diminta maka terdapat 3 kasus, yaitu:

- Apabila Persentase hasil $>$ compression rate maka nilai $\text{minThreshold} = \text{mid}$ untuk selain metode SSIM dan $\text{maxThreshold} = \text{mid}$ untuk metode SSIM. Perubahan rentang threshold dimaksudkan agar gambar semakin terkompresi
- Apabila Persentase hasil $<$ compression rate maka nilai $\text{maxThreshold} = \text{mid}$ untuk selain metode SSIM dan $\text{minThreshold} = \text{mid}$ untuk metode SSIM. Perubahan rentang threshold dimaksudkan agar gambar semakin bagus
- Apabila hingga $\text{minThreshold} = \text{maxThreshold}$ tidak ditemukan threshold maka pencarian dihentikan dan akan digunakan threshold yang terakhir kali digunakan.

Untuk bonus ini penulis mengatur minBlocksize selalu 2

3. GIF

Untuk membuat GIF cukup sederhana kita membutuhkan beberapa frame untuk membuat animasi. Frame di-*generate* sebanyak kedalaman quadtree.

G. Link Repository

Berikut adalah link repository:

https://github.com/AryoBama/Tucil2_13523088

H. Checklist

| 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 | ✓ | |