

LAPORAN TUGAS KECIL II
IF2211 STRATEGI ALGORITMA
Penyelesaian **Kompresi Gambar** dengan **Metode Quadtree**



Disusun oleh:
Benedict Presley 13523067
Steven Owen Liauw 13523103

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

Daftar Isi

BAGIAN I ALGORITMA DIVIDE CONQUER.....
BAB II SOURCE PROGRAM.....
BAGIAN III SCREENSHOT HASIL TEST.....
BAGIAN IV ANALISIS.....
BAGIAN V BONUS.....
LINK REPOSITORY.....
CHECKLIST.....

BAGIAN I

Algoritma Divide and Conquer

Algoritma *divide and conquer* adalah teknik pemecahan masalah dengan cara membagi masalah besar menjadi beberapa submasalah yang lebih kecil, menyelesaikan masing-masing submasalah secara rekursif, lalu menggabungkan hasilnya untuk memperoleh solusi akhir. Pada kompresi gambar dengan metode QuadTree algoritma *Divide and Conquer* memiliki pendekatan :

- **Divide :**
 - Gambar persegi dibagi menjadi empat bagian kuadran yang sama besar: kiri-atas, kanan-atas, kiri-bawah, kanan-bawah.
 - Proses ini dilakukan rekursif untuk setiap bagian, sehingga menghasilkan struktur pohon di mana tiap simpul bisa memiliki empat anak.
- **Conquer :**
 - Untuk setiap blok/kuadran, dicek apakah semua pikselnya cukup homogen, dengan metrik yang ditentukan (Variance / MAD / Max Pixel Difference / Entropy / SSIM).
 - Jika suatu blok homogen, tidak perlu dibagi lagi. Blok itu disimpan sebagai satu simpul daun.
 - Jika tidak homogen, blok akan dibagi lagi menjadi 4, dan proses ini diulang (rekursif).
- **Combine :**
 - Setelah seluruh gambar diproses, semua blok yang sudah cukup homogen menjadi simpul daun.
 - Simpul daun tersebut yang kemudian digunakan untuk rekonstruksi kembali.

Pseudo Code Algoritma *Divide and Conquer* :

```
Fungsi BuildQuadTree(image: array of RGBPixel, x, y, w, h: integer,  
threshold: real, minBlockSize: integer,  
errorMethod: integer, imageWidth: integer) → QuadTreeNode
```

Deklarasi:

```
avgColor: RGBPixel  
error: real  
halfWidth, remWidth, halfHeight, remHeight: integer  
node: QuadTreeNode
```

Algoritma:

```
avgColor ← CalculateAverageColor(image, x, y, w, h, imageWidth)  
  
switch errorMethod do  
    case 1:  
        error ← CalculateVariance(image, x, y, w, h, avgColor, imageWidth)  
    case 2:  
        error ← CalculateMeanAbsoluteDeviation(image, x, y, w, h, avgColor, imageWidth)  
    case 3:  
        error ← CalculateMaxPixelDifference(image, x, y, w, h, avgColor, imageWidth)  
    case 4:  
        error ← CalculateEntropy(image, x, y, w, h, avgColor, imageWidth)  
    case 5:  
        ssim ← CalculateSSIM(image, x, y, w, h, avgColor, imageWidth)  
        error ← 1.0 - ssim  
endswitch  
  
if error < threshold or (w × h) < minBlockSize then  
    return Node(x, y, w, h, avgColor, isLeaf = true)  
endif
```

```

halfWidth ← w div 2
remWidth ← w - halfWidth
halfHeight ← h div 2
remHeight ← h - halfHeight

node ← Node(x, y, w, h, avgColor, isLeaf = false)

node.atasKiri ← BuildQuadTree(image, x, y, halfWidth, halfHeight, threshold, minBlockSize, errorMethod, imageWidth)
node.atasKanan ← BuildQuadTree(image, x + halfWidth, y, remWidth, halfHeight, threshold, minBlockSize, errorMethod, imageWidth)
node.bawahKiri ← BuildQuadTree(image, x, y + halfHeight, halfWidth, remHeight, threshold, minBlockSize, errorMethod, imageWidth)
node.bawahKanan ← BuildQuadTree(image, x + halfWidth, y + halfHeight, remWidth, remHeight, threshold, minBlockSize,
errorMethod, imageWidth)

return node

```

Prosedur reconstructImage(outputImage: array of RGBPixel,
 node: QuadTreeNode,
 imageWidth: integer)

Deklarasi:

i, j, idx: integer

Algoritma:

```

if node = null then
    return
endif
```

```

if node.isLeaf then
    for i ← 0 to node.height - 1 do
        for j ← 0 to node.width - 1 do
            idx ← (node.y + i) × imageWidth + (node.x + j)
            outputImage[idx] ← node.color
        endfor
    endfor
    return
endif
```

```

reconstructImage(outputImage, node.atasKiri, imageWidth)
reconstructImage(outputImage, node.atasKanan, imageWidth)
reconstructImage(outputImage, node.bawahKiri, imageWidth)
reconstructImage(outputImage, node.bawahKanan, imageWidth)
```


BAB II

SOURCE PROGRAM

Projek ini ditulis dalam Bahasa C++, menggunakan *library*:

1. stb_image
2. iff2gif

Di program ini, setiap file memiliki fungsinya masing masing yaitu:

- ImageProcessing.cpp : program utama yang melakukan kalkulasi dan menghasilkan solusi.
- Metrics.cpp : program yang berisi metric penentu homogenitas.
- QuadTree.cpp : implementasi dari struktur data QuadTree.
- gifenc.c :implementasi dari bonus GIF.

Berikut *sourcecode*-nya:

ImageProcessing.cpp

```
#define STB_IMAGE_IMPLEMENTATION
#include "stb_image.h"

#define STB_IMAGE_WRITE_IMPLEMENTATION
#include "stb_image_write.h"

#include "gifenc.h"
#include "QuadTree.hpp"
#include "Metrics.hpp"
#include "ImageLoadException.hpp"

#include "gif-library/iff2gif/neuquant.hpp"

#include <iostream>
#include <iomanip>
#include <sys/stat.h>
#include <chrono>
#include <fstream>
#include <string>

std::vector<RGBPixel> LoadImage(std::string fileName, int &width, int &height) {
    int channels;
    unsigned char* image_data = stbi_load(fileName.c_str(), &width, &height, &channels, 0);

    if (!image_data) {
        throw ImageLoadException("Can't open file: " + fileName);
    }
```

```

std::vector<RGBPixel> pixels;

for (int i = 0; i < height; i++) {
    for (int j = 0; j < width; j++) {
        int index = (i * width + j) * channels;

        uint8_t r = image_data[index + 0];
        uint8_t g = image_data[index + 1];
        uint8_t b = image_data[index + 2];

        pixels.emplace_back(RGBPixel(r, g, b));
    }
}

return pixels;
}

void SaveImage(std::string fileName, const std::vector<RGBPixel> &image, int &width, int &height, bool show) {
    std::vector<uint8_t> rawData;
    rawData.reserve(width * height * 3);

    for (const auto& pixel : image) {
        rawData.push_back(pixel.r);
        rawData.push_back(pixel.g);
        rawData.push_back(pixel.b);
    }

    std::string ext = fileName.substr(fileName.find_last_of('.') + 1);
    for (int i = 0; ext[i] != '\0'; ++i) {
        if ('A' <= ext[i] && ext[i] <= 'Z') {
            ext[i] += 32;
        }
    }

    bool success = false;

    if (ext == "png") {
        success = stbi_write_png(fileName.c_str(), width, height, 3, rawData.data(), width * 3);
    }
    else if (ext == "jpg" || ext == "jpeg") {
        int quality = 100;
        success = stbi_write_jpg(fileName.c_str(), width, height, 3, rawData.data(), quality);
    }
    else {
        std::cerr << "Unsupported file extension: ." << ext << std::endl;
        return;
    }
}

```

```

        if(show) {
            if (success) {
                std::cout << "Gambar berhasil disimpan di " << fileName << std::endl;
            } else {
                std::cerr << "Gambar tidak berhasil disimpan" << std::endl;
            }
        }
    }

RGBPixel CalculateAverageColor(const std::vector<RGBPixel> &image, int x, int y, int width,
int height, int &imageWidth) {
    uint32_t r = 0, g = 0, b = 0;
    for (int i = y; i < y + height; i++)
    {
        for (int j = x; j < x + width; j++)
        {
            int idx = i * imageWidth + j;
            r += image[idx].r;
            g += image[idx].g;
            b += image[idx].b;
        }
    }
    int totalPixel = width * height;
    return RGBPixel((uint8_t)(r / totalPixel), (uint8_t)(g / totalPixel), (uint8_t)(b / totalPixel));
}

std::unique_ptr<QuadTreeNode> BuildQuadTree(std::vector<RGBPixel> &image, int x, int y, int w,
int h, double threshold, int minBlockSize, int errorMeasurementChoice, int imageWidth) {
    if(w <= 0 || h <= 0) {
        return nullptr;
    }

    RGBPixel avgColor = CalculateAverageColor(image, x, y, w, h, imageWidth);
    double error;

    switch (errorMeasurementChoice)
    {
    case 1:
        // Variance
        error = CalculateVariance(image, x, y, w, h, avgColor, imageWidth);
        break;
    case 2:
        // Mean Absolute Deviation (MAD)
        error = CalculateMeanAbsoluteDeviation(image, x, y, w, h, avgColor, imageWidth);
        break;
    case 3:
        // Max Pixel Difference
        error = CalculateMaxPixelDifference(image, x, y, w, h, avgColor, imageWidth);
    }
}

```

```

break;

case 4:
    // Entropy
    error = CalculateEntropy(image, x, y, w, h, avgColor, imageWidth);
    break;

case 5:
    // SSIM
    double ssim = CalculateSSIM(image, x, y, w, h, avgColor, imageWidth);
    error = 1.0 - ssim;
    break;
}

if (error < threshold || (w * h) < minBlockSize)
{
    return std::make_unique<QuadTreeNode>(x, y, w, h, avgColor, true);
}

int halfWidth = w / 2;
int remWidth = w - halfWidth;
int halfHeight = h / 2;
int remHeight = h - halfHeight;

auto node = std::make_unique<QuadTreeNode>(x, y, w, h, avgColor, false);
node->atasKiri = BuildQuadTree(image, x, y, halfWidth, halfHeight, threshold,
minBlockSize, errorMeasurementChoice, imageWidth);
node->atasKanan = BuildQuadTree(image, x + halfWidth, y, remWidth, halfHeight, threshold,
minBlockSize, errorMeasurementChoice, imageWidth);
node->bawahKiri = BuildQuadTree(image, x, y + halfHeight, halfWidth, remHeight, threshold,
minBlockSize, errorMeasurementChoice, imageWidth);
node->bawahKanan = BuildQuadTree(image, x + halfWidth, y + halfHeight, remWidth,
remHeight, threshold, minBlockSize, errorMeasurementChoice, imageWidth);
return node;
}

void reconstructImage(std::vector<RGBPixel> &outputImage, std::unique_ptr<QuadTreeNode> &node,
int &imageWidth) {
    if (!node)
    {
        return;
    }

    if (node->isLeaf)
    {
        for (int i = 0; i < node->height; i++)
        {
            for (int j = 0; j < node->width; j++)
            {
                int idx = (node->y + i) * imageWidth + (node->x + j);
                outputImage[idx] = node->color;
            }
        }
    }
}

```

```

        outputImage[idx] = node->color;
    }
}
return;
}

reconstructImage(outputImage, node->atasKiri, imageWidth);
reconstructImage(outputImage, node->atasKanan, imageWidth);
reconstructImage(outputImage, node->bawahKiri, imageWidth);
reconstructImage(outputImage, node->bawahKanan, imageWidth);
}

double CalculateCompressionRatio(const std::string &uncompressedFile, const std::string &compressedFile, bool show) {
    struct stat uncompressedStat, compressedStat;
    if (stat(uncompressedFile.c_str(), &uncompressedStat) != 0) {
        throw ImageLoadException("Cannot get file size: " + uncompressedFile);
    }
    if (stat(compressedFile.c_str(), &compressedStat) != 0) {
        throw ImageLoadException("Cannot get file size: " + compressedFile);
    }

    double uncompressedSize = uncompressedStat.st_size;
    double compressedSize = compressedStat.st_size;
    if (show) {
        std::cout << "Ukuran Sebelum Kompresi: " << uncompressedSize << " bytes" << std::endl;
        std::cout << "Ukuran Setelah Kompresi: " << compressedSize << " bytes" << std::endl;
    }
    return (1 - (compressedSize / uncompressedSize)) * 100.0;
}

int GetMaxDepth(std::unique_ptr<QuadTreeNode> &node) {
    if (!node) {
        return 0;
    }

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

    return std::max({GetMaxDepth(node->atasKiri),
                    GetMaxDepth(node->atasKiri),
                    GetMaxDepth(node->bawahKiri),
                    GetMaxDepth(node->bawahKanan)}) + 1;
}

int GetNodeCount(std::unique_ptr<QuadTreeNode> &node) {
    if (!node) {
        return 0;
    }
}
```

```

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

return GetNodeCount(node->atasKiri) +
    GetNodeCount(node->atasKanan) +
    GetNodeCount(node->bawahKiri) +
    GetNodeCount(node->bawahKanan) + 1;
}

void SaveGif(const std::string &gifOutputPath, const std::vector<RGBPixel> &image,
std::unique_ptr<QuadTreeNode> &root, int imageWidth, int imageHeight) {
    std::vector<uint8_t> rgbData(image.size() * 3);
    for (size_t i = 0; i < image.size(); ++i) {
        rgbData[i * 3 + 0] = image[i].r;
        rgbData[i * 3 + 1] = image[i].g;
        rgbData[i * 3 + 2] = image[i].b;
    }

    NeuQuant* neuquant = new NeuQuant(256);
    Quantizer* quantizer = neuquant;
    Palette palette;

    uint8_t gifPalette[256 * 3];
    memset(gifPalette, 0, sizeof(gifPalette));

    if((int)image.size() > 256) {
        quantizer->AddPixels(rgbData.data(), image.size());
        Palette palette = quantizer->GetPalette();
        for (size_t i = 0; i < std::min(palette.size(), size_t(256)); ++i) {
            gifPalette[i * 3 + 0] = palette[i].red;
            gifPalette[i * 3 + 1] = palette[i].green;
            gifPalette[i * 3 + 2] = palette[i].blue;
        }
    }
    else {
        for (size_t i = 0; i < image.size(); ++i) {
            gifPalette[i * 3 + 0] = image[i].r;
            gifPalette[i * 3 + 1] = image[i].g;
            gifPalette[i * 3 + 2] = image[i].b;
        }
    }

    ge_GIF* gif = ge_new_gif(gifOutputPath.c_str(), imageWidth, imageHeight, gifPalette, 8, 0,
0);

    if (!gif) {
        std::cerr << "Failed to create GIF." << std::endl;
        delete quantizer;
    }
}

```

```

        return;
    }

std::vector<QuadTreeNode*> nodes, newNodes;
nodes.push_back(root.get());

int counter = 0;
bool cont = true;

while (cont && !nodes.empty()) {
    cont = false;
    std::vector<uint8_t> frameRGB(imageWidth * imageHeight * 3);
    std::vector<uint8_t> frameIndexed(imageWidth * imageHeight);

    for (QuadTreeNode* node : nodes) {
        if (!node) continue;

        for (int i = 0; i < node->height; i++) {
            for (int j = 0; j < node->width; j++) {
                int xx = node->x + j, yy = node->y + i;
                int idx = yy * imageWidth + xx;

                frameRGB[idx * 3 + 0] = node->color.r;
                frameRGB[idx * 3 + 1] = node->color.g;
                frameRGB[idx * 3 + 2] = node->color.b;

                ColorRegister pixel = {node->color.r, node->color.g, node->color.b};
                frameIndexed[idx] = neuquant->lookup(pixel);
            }
        }
    }

    if (node->isLeaf) {
        newNodes.push_back(node);
    }
    else {
        cont = true;
        newNodes.push_back(node->atasKiri.get());
        newNodes.push_back(node->atasKanan.get());
        newNodes.push_back(node->bawahKiri.get());
        newNodes.push_back(node->bawahKanan.get());
    }
}

memcpy(gif->frame, frameIndexed.data(), imageWidth * imageHeight);

ge_add_frame(gif, 100);

nodes.swap(newNodes);
std::vector<QuadTreeNode*>().swap(newNodes);

```

```

}

std::vector<QuadTreeNode*>().swap(nodes);
std::vector<QuadTreeNode*>().swap(newNodes);
delete quantizer;

if (gif) {
    ge_close_gif(gif);
    std::cout << "GIF berhasil disimpan di " << gifOutputPath << std::endl;
}
}

int main() {
    try {
        int width, height;

        std::string originalImagePath;
        std::string compressedImagePath;
        std::string gifOutputPath;
        int errorMeasurementChoice;
        double threshold;
        int minBlockSize;
        double targetCompressionRatio;
        double low, high;

        std::cout << "Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): ";
        std::getline(std::cin, originalImagePath);

        struct stat buffer;
        while (stat(originalImagePath.c_str(), &buffer) != 0) {
            std::cerr << "File tidak ditemukan: " << originalImagePath << std::endl;
            std::cout << "Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): ";
            std::getline(std::cin, originalImagePath);
        }

        std::cout << "Pilih metode perhitungan error:\n";
        std::cout << "1. Variansi\n";
        std::cout << "2. Rata-rata Deviasi Absolut (MAD)\n";
        std::cout << "3. Selisih Piksel Maksimum\n";
        std::cout << "4. Entropi\n";
        std::cout << "5. SSIM (Structural Similarity Index)\n";
        std::cout << "Masukkan nomor metode (1-5): ";
        std::cin >> errorMeasurementChoice;

        while (errorMeasurementChoice < 1 || errorMeasurementChoice > 5) {
            std::cout << "Pilihan tidak valid. Silakan pilih antara 1-5: ";
            std::cin >> errorMeasurementChoice;
        }

        switch (errorMeasurementChoice)

```

```

{
case 1:
    // Variance
    low = 0; high = 16256.25;
    break;
case 2:
    // Mean Absolute Deviation (MAD)
    low = 0; high = 127.5;
    break;
case 3:
    // Max Pixel Difference
    low = 0; high = 255;
    break;

case 4:
    // Entropy (rentang 0-8)
    low = 0; high = 8;
    break;

case 5:
    // SSIM (rentang 0-1)
    low = -1; high = 1;
    break;
}

std::cout << "Masukkan nilai threshold: ";
std::cin >> threshold;

while (threshold < low || threshold > high) {
    std::cout << "Nilai threshold tidak valid. Silakan masukkan nilai antara " << low
<< " dan " << high << ":" ;
    std::cin >> threshold;
}

std::cout << "Masukkan ukuran blok minimum: ";
std::cin >> minBlockSize;

while (minBlockSize < 1) {
    std::cout << "Ukuran blok minimum tidak valid. Silakan masukkan nilai lebih besar
dari 0: ";
    std::cin >> minBlockSize;
}

std::cout << "Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0
untuk 100%): ";
std::cin >> targetCompressionRatio;
std::cin.ignore();

while (targetCompressionRatio < 0.0 || targetCompressionRatio > 1.0) {
    std::cout << "Rasio kompresi tidak valid. Silakan masukkan nilai antara 0.0 dan

```

```
1.0: ";
    std::cin >> targetCompressionRatio;
    std::cin.ignore();
}

std::cout << "Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): ";
std::getline(std::cin, compressedImagePath);

std::cout << "Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): ";
std::getline(std::cin, gif outputPath);

std::cout << "Memproses gambar..." << std::endl;

auto startTime = std::chrono::high_resolution_clock::now();

std::unique_ptr<QuadTreeNode> root;
std::vector<RGBPixel> image = LoadImage(originalImagePath, width, height);

std::vector<RGBPixel> outputImage(width * height);

if (targetCompressionRatio == 0.0) {
    root = BuildQuadTree(image, 0, 0, width, height, threshold, minBlockSize,
errorMeasurementChoice, width);
}
else {
    // Mengasumsikan rasio bergantung sepenuhnya pada threshold

    targetCompressionRatio *= 100.0;

    int tempBlockSize = 1;
    long double L = low, R = high;

    for (int _ = 0; _ < 20; _++) {
        long double M = (L + R) / 2.0;

        root = BuildQuadTree(image, 0, 0, width, height, M, tempBlockSize,
errorMeasurementChoice, width);

        outputImage = std::vector<RGBPixel>(width * height);
        reconstructImage(outputImage, root, width);
        SaveImage(compressedImagePath, outputImage, width, height, false);
        double compressionRatio = CalculateCompressionRatio(originalImagePath,
compressedImagePath, false);
        if (compressionRatio < targetCompressionRatio) {
            L = M;
        }
        else {
            R = M;
        }
    }
}
```

```

        }
    }

    reconstructImage(outputImage, root, width);

    SaveImage(compressedImagePath, outputImage, width, height, true);
    SaveGif(gifOutputPath, image, root, width, height);

    auto endTime = std::chrono::high_resolution_clock::now();
    auto duration = std::chrono::duration_cast<std::chrono::milliseconds>(endTime - startTime).count();
    std::cout << "Waktu pemrosesan: " << duration << " ms" << std::endl;

    double compressionRatio = CalculateCompressionRatio(originalImagePath,
compressedImagePath, 1);
    std::cout << std::fixed << std::setprecision(6) << "Rasio Kompresi: " <<
compressionRatio << "%" << std::endl;

    int maxDepth = GetMaxDepth(root);
    std::cout << "Kedalaman Maksimum: " << maxDepth << std::endl;

    int nodeCount = GetNodeCount(root);
    std::cout << "Banyak Simpul: " << nodeCount << std::endl;
}

catch (const ImageLoadException &e) {
    std::cerr << "Error: " << e.what() << std::endl;
}

return 0;
}

```

Metrics.cpp

```

#include "Metrics.hpp"

double CalculateVariance(const std::vector<RGBPixel> &image, int x, int y, int width, int
height, const RGBPixel &avgColor, int &imageWidth) {
    double variance = 0.0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;
            variance += (image[idx].r - avgColor.r) * (image[idx].r - avgColor.r) +
(image[idx].r - avgColor.g) * (image[idx].r - avgColor.g) +
(image[idx].r - avgColor.b) * (image[idx].r - avgColor.b);
        }
    }
}

```

```

        }
    }

    return variance / (double)(3 * width * height);
}

double CalculateMeanAbsoluteDeviation(const std::vector<RGBPixel> &image, int x, int y, int
width, int height, const RGBPixel &avgColor, int &imageWidth) {
    double mad = 0.0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;
            mad += abs(image[idx].r - avgColor.r) +
                  abs(image[idx].g - avgColor.g) +
                  abs(image[idx].b - avgColor.b);
        }
    }

    return mad / (double)(3 * width * height);
}

double CalculateMaxPixelDifference(const std::vector<RGBPixel> &image, int x, int y, int
width, int height, const RGBPixel &avgColor, int &imageWidth) {
    uint8_t minR = 255, maxR = 0;
    uint8_t minG = 255, maxG = 0;
    uint8_t minB = 255, maxB = 0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;

            minR = std::min(minR, image[idx].r);
            maxR = std::max(maxR, image[idx].r);

            minG = std::min(minG, image[idx].g);
            maxG = std::max(maxG, image[idx].g);

            minB = std::min(minB, image[idx].b);
            maxB = std::max(maxB, image[idx].b);
        }
    }

    double diffR = (double)(maxR - minR);
    double diffG = (double)(maxG - minG);
    double diffB = (double)(maxB - minB);
    double diffRGB = (diffR + diffG + diffB) / 3.0;
    return diffRGB;
}

```

```

double CalculateEntropy(const std::vector<RGBPixel> &image, int x, int y, int width, int height, const RGBPixel &avgColor, int &imageWidth) {
    std::vector<int> freqR(256, 0), freqG(256, 0), freqB(256, 0);
    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;
            freqR[image[idx].r]++;
            freqG[image[idx].g]++;
            freqB[image[idx].b]++;
        }
    }

    int totalPixels = width * height;
    double H = 0.0;
    for (int i = 0; i < 256; i++) {
        if (freqR[i] > 0) {
            double p = (double)freqR[i] / (double)totalPixels;
            H -= p * log2(p);
        }
        if (freqG[i] > 0) {
            double p = (double)freqG[i] / (double)totalPixels;
            H -= p * log2(p);
        }
        if (freqB[i] > 0) {
            double p = (double)freqB[i] / (double)totalPixels;
            H -= p * log2(p);
        }
    }

    return H / 3.0;
}

double CalculateSSIM(const std::vector<RGBPixel> &image, int x, int y, int width, int height, const RGBPixel &avgColor, int &imageWidth) {
    int totalPixels = width * height;
    double sum1R = 0.0, sum1R2 = 0.0, sum12R = 0.0;
    double sum1G = 0.0, sum1G2 = 0.0, sum12G = 0.0;
    double sum1B = 0.0, sum1B2 = 0.0, sum12B = 0.0;

    double mean2R = (double)avgColor.r;
    double mean2G = (double)avgColor.g;
    double mean2B = (double)avgColor.b;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;
            double r1 = (double)image[idx].r;
            sum1R += r1;
            sum1R2 += r1 * r1;
            sum12R += r1 * mean2R;
        }
    }

    double mean1R = sum1R / totalPixels;
    double mean1R2 = sum1R2 / totalPixels;
    double mean12R = sum12R / totalPixels;

    double sum2R = 0.0, sum2G = 0.0, sum2B = 0.0;
    double sum2R2 = 0.0, sum2G2 = 0.0, sum2B2 = 0.0;
    double sum21R = 0.0, sum21G = 0.0, sum21B = 0.0;
    double sum21R2 = 0.0, sum21G2 = 0.0, sum21B2 = 0.0;
    double sum212R = 0.0, sum212G = 0.0, sum212B = 0.0;

    for (int i = y; i < y + height; i++) {
        for (int j = x; j < x + width; j++) {
            int idx = i * imageWidth + j;
            double r2 = (double)image[idx].r;
            sum2R += r2;
            sum2R2 += r2 * r2;
            sum21R += r1 * r2;
            sum21R2 += r1 * r2 * r2;
            sum212R += r1 * r2 * mean2R;
        }
    }

    double mean2R = sum2R / totalPixels;
    double mean2R2 = sum2R2 / totalPixels;
    double mean21R = sum21R / totalPixels;
    double mean21R2 = sum21R2 / totalPixels;
    double mean212R = sum212R / totalPixels;

    double numer = 2 * mean21R - mean1R * mean2R;
    double denom = sqrt((mean1R * mean1R) * (mean2R * mean2R));
    double ssim = numer / denom;
}

```

```

        double g1 = (double)image[idx].g;
        sum1G += g1;
        sum1G2 += g1 * g1;
        sum12G += g1 * mean2G;

        double b1 = (double)image[idx].b;
        sum1B += b1;
        sum1B2 += b1 * b1;
        sum12B += b1 * mean2B;
    }
}

double mean1R = sum1R / (double)totalPixels;
double mean1G = sum1G / (double)totalPixels;
double mean1B = sum1B / (double)totalPixels;

double var1R = sum1R2 / (double)totalPixels - mean1R * mean1R;
double var1G = sum1G2 / (double)totalPixels - mean1G * mean1G;
double var1B = sum1B2 / (double)totalPixels - mean1B * mean1B;

double covR = sum12R / (double)totalPixels - mean1R * mean2R;
double covG = sum12G / (double)totalPixels - mean1G * mean2G;
double covB = sum12B / (double)totalPixels - mean1B * mean2B;

double L_val = 255.0;
double K1 = 0.01, K2 = 0.03;
double C1 = (K1 * L_val) * (K1 * L_val);
double C2 = (K2 * L_val) * (K2 * L_val);

double ssimR = (2 * mean1R * mean2R + C1) * (2 * covR + C2) / ((mean1R * mean1R + mean2R * mean2R + C1) * (var1R + C2));
double ssimG = (2 * mean1G * mean2G + C1) * (2 * covG + C2) / ((mean1G * mean1G + mean2G * mean2G + C1) * (var1G + C2));
double ssimB = (2 * mean1B * mean2B + C1) * (2 * covB + C2) / ((mean1B * mean1B + mean2B * mean2B + C1) * (var1B + C2));

return (ssimR + ssimG + ssimB) / 3.0;
}

```

QuadTree.cpp

```

#include "QuadTree.hpp"

RGBPixel::RGBPixel() : r(0), g(0), b(0) {}
RGBPixel::RGBPixel(uint8_t r, uint8_t g, uint8_t b) : r(r), g(g), b(b) {}

```

```

QuadTreeNode::QuadTreeNode(int x, int y, int width, int height, RGBPixel color, bool
isLeaf)
: x(x), y(y), width(width), height(height), color(color), isLeaf(isLeaf),
atasKiri(nullptr), atasKanan(nullptr), bawahKiri(nullptr), bawahKanan(nullptr)
{}

```

gifenc.c

```

#include "gifenc.h"

#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <sys/types.h>
#include <sys/stat.h>
#include <fcntl.h>
#ifndef _WIN32
#include <io.h>
#else
#include <unistd.h>
#endif

/* helper to write a little-endian 16-bit number portably */
#define write_num(fd, n) write((fd), (uint8_t []) {(n) & 0xFF, (n) >> 8}, 2)

static uint8_t vga[0x30] = {
    0x00, 0x00, 0x00,
    0xAA, 0x00, 0x00,
    0x00, 0xAA, 0x00,
    0xAA, 0x55, 0x00,
    0x00, 0x00, 0xAA,
    0xAA, 0x00, 0xAA,
    0x00, 0xAA, 0xAA,
    0xAA, 0xAA, 0xAA,
    0x55, 0x55, 0x55,
    0xFF, 0x55, 0x55,
    0x55, 0xFF, 0x55,
    0xFF, 0xFF, 0x55,
    0x55, 0x55, 0xFF,
    0xFF, 0x55, 0xFF,
    0x55, 0xFF, 0xFF,
    0xFF, 0xFF, 0xFF,
};

struct Node {
    uint16_t key;

```

```

        struct Node *children[];
};

typedef struct Node Node;

static Node *
new_node(uint16_t key, int degree)
{
    Node *node = calloc(1, sizeof(*node) + degree * sizeof(Node *));
    if (node)
        node->key = key;
    return node;
}

static Node *
new_trie(int degree, int *nkeys)
{
    Node *root = new_node(0, degree);
    /* Create nodes for single pixels. */
    for (*nkeys = 0; *nkeys < degree; (*nkeys)++)
        root->children[*nkeys] = new_node(*nkeys, degree);
    *nkeys += 2; /* skip clear code and stop code */
    return root;
}

static void
del_trie(Node *root, int degree)
{
    if (!root)
        return;
    for (int i = 0; i < degree; i++)
        del_trie(root->children[i], degree);
    free(root);
}

#define write_and_store(s, dst, fd, src, n) \
do { \
    write(fd, src, n); \
    if (s) { \
        memcpy(dst, src, n); \
        dst += n; \
    } \
} while (0);

static void put_loop(ge_GIF *gif, uint16_t loop);

ge_GIF *
ge_new_gif(
    const char *fname, uint16_t width, uint16_t height,
    uint8_t *palette, int depth, int bgindex, int loop
)

```

```

{
    int i, r, g, b, v;
    int store_gct, custom_gct;
    int nbuffers = bgindex < 0 ? 2 : 1;
    ge_GIF *gif = calloc(1, sizeof(*gif) + nbuffers*width*height);
    if (!gif)
        goto no_gif;
    gif->w = width; gif->h = height;
    gif->bgindex = bgindex;
    gif->frame = (uint8_t *) &gif[1];
    gif->back = &gif->frame[width*height];
#endif _WIN32
    gif->fd = creat(fname, S_IWRITE);
#else
    gif->fd = creat(fname, 0666);
#endif
    if (gif->fd == -1)
        goto no_fd;
#endif _WIN32
    setmode(gif->fd, O_BINARY);
#endif
    write(gif->fd, "GIF89a", 6);
    write_num(gif->fd, width);
    write_num(gif->fd, height);
    store_gct = custom_gct = 0;
    if (palette) {
        if (depth < 0)
            store_gct = 1;
        else
            custom_gct = 1;
    }
    if (depth < 0)
        depth = -depth;
    gif->depth = depth > 1 ? depth : 2;
    write(gif->fd, (uint8_t []) {0xF0 | (depth-1), (uint8_t) bgindex, 0x00}, 3);
    if (custom_gct) {
        write(gif->fd, palette, 3 << depth);
    } else if (depth <= 4) {
        write_and_store(store_gct, palette, gif->fd, vga, 3 << depth);
    } else {
        write_and_store(store_gct, palette, gif->fd, vga, sizeof(vga));
        i = 0x10;
        for (r = 0; r < 6; r++) {
            for (g = 0; g < 6; g++) {
                for (b = 0; b < 6; b++) {
                    write_and_store(store_gct, palette, gif->fd,
                        ((uint8_t []) {r*51, g*51, b*51}), 3
                    );
                    if (++i == 1 << depth)
                        goto done_gct;
                }
            }
        }
    }
}

```

```

        }
    }
}

for (i = 1; i <= 24; i++) {
    v = i * 0xFF / 25;
    write_and_store(store_gct, palette, gif->fd,
        ((uint8_t []) {v, v, v}), 3
    );
}
}

done_gct:
if (loop >= 0 && loop <= 0xFFFF)
    put_loop(gif, (uint16_t) loop);
return gif;
no_fd:
free(gif);
no_gif:
return NULL;
}

static void
put_loop(ge_GIF *gif, uint16_t loop)
{
    write(gif->fd, (uint8_t []) {'!', 0xFF, 0x0B}, 3);
    write(gif->fd, "NETSCAPE2.0", 11);
    write(gif->fd, (uint8_t []) {0x03, 0x01}, 2);
    write_num(gif->fd, loop);
    write(gif->fd, "\0", 1);
}

/* Add packed key to buffer, updating offset and partial.
 *   gif->offset holds position to put next *bit*
 *   gif->partial holds bits to include in next byte */
static void
put_key(ge_GIF *gif, uint16_t key, int key_size)
{
    int byte_offset, bit_offset, bits_to_write;
    byte_offset = gif->offset / 8;
    bit_offset = gif->offset % 8;
    gif->partial |= ((uint32_t) key) << bit_offset;
    bits_to_write = bit_offset + key_size;
    while (bits_to_write >= 8) {
        gif->buffer[byte_offset++] = gif->partial & 0xFF;
        if (byte_offset == 0xFF) {
            write(gif->fd, "\xFF", 1);
            write(gif->fd, gif->buffer, 0xFF);
            byte_offset = 0;
        }
        gif->partial >>= 8;
        bits_to_write -= 8;
    }
}

```

```

    }
    gif->offset = (gif->offset + key_size) % (0xFF * 8);
}

static void
end_key(ge_GIF *gif)
{
    int byte_offset;
    byte_offset = gif->offset / 8;
    if (gif->offset % 8)
        gif->buffer[byte_offset++] = gif->partial & 0xFF;
    if (byte_offset) {
        write(gif->fd, (uint8_t []) {byte_offset}, 1);
        write(gif->fd, gif->buffer, byte_offset);
    }
    write(gif->fd, "\0", 1);
    gif->offset = gif->partial = 0;
}

static void
put_image(ge_GIF *gif, uint16_t w, uint16_t h, uint16_t x, uint16_t y)
{
    int nkeys, key_size, i, j;
    Node *node, *child, *root;
    int degree = 1 << gif->depth;

    write(gif->fd, ",", 1);
    write_num(gif->fd, x);
    write_num(gif->fd, y);
    write_num(gif->fd, w);
    write_num(gif->fd, h);
    write(gif->fd, (uint8_t []) {0x00, gif->depth}, 2);
    root = node = new_trie(degree, &nkeys);
    key_size = gif->depth + 1;
    put_key(gif, degree, key_size); /* clear code */
    for (i = y; i < y+h; i++) {
        for (j = x; j < x+w; j++) {
            uint8_t pixel = gif->frame[i*gif->w+j] & (degree - 1);
            child = node->children[pixel];
            if (child) {
                node = child;
            } else {
                put_key(gif, node->key, key_size);
                if (nkeys < 0x1000) {
                    if (nkeys == (1 << key_size))
                        key_size++;
                    node->children[pixel] = new_node(nkeys++, degree);
                } else {
                    put_key(gif, degree, key_size); /* clear code */
                    del_trie(root, degree);
                }
            }
        }
    }
}

```

```

        root = node = new_trie(degree, &nkeys);
        key_size = gif->depth + 1;
    }
    node = root->children[pixel];
}
}

put_key(gif, node->key, key_size);
put_key(gif, degree + 1, key_size); /* stop code */
end_key(gif);
del_trie(root, degree);
}

static int
get_bbox(ge_GIF *gif, uint16_t *w, uint16_t *h, uint16_t *x, uint16_t *y)
{
    int i, j, k;
    int left, right, top, bottom;
    uint8_t back;
    left = gif->w; right = 0;
    top = gif->h; bottom = 0;
    k = 0;
    for (i = 0; i < gif->h; i++) {
        for (j = 0; j < gif->w; j++, k++) {
            back = gif->bgindex >= 0 ? gif->bgindex : gif->back[k];
            if (gif->frame[k] != back) {
                if (j < left)    left    = j;
                if (j > right)   right   = j;
                if (i < top)     top     = i;
                if (i > bottom)  bottom  = i;
            }
        }
    }
    if (left != gif->w && top != gif->h) {
        *x = left; *y = top;
        *w = right - left + 1;
        *h = bottom - top + 1;
        return 1;
    } else {
        return 0;
    }
}

static void
add_graphics_control_extension(ge_GIF *gif, uint16_t d)
{
    uint8_t flags = ((gif->bgindex >= 0 ? 2 : 1) << 2) + 1;
    write(gif->fd, (uint8_t []) {'!', 0xF9, 0x04, flags}, 4);
    write_num(gif->fd, d);
    write(gif->fd, (uint8_t []) {(uint8_t) gif->bgindex, 0x00}, 2);
}

```

```

}

void
ge_add_frame(ge_GIF *gif, uint16_t delay)
{
    uint16_t w, h, x, y;
    uint8_t *tmp;

    if (delay || (gif->bgindex >= 0))
        add_graphics_control_extension(gif, delay);
    if (gif->nframes == 0) {
        w = gif->w;
        h = gif->h;
        x = y = 0;
    } else if (!get_bbox(gif, &w, &h, &x, &y)) {
        /* image's not changed; save one pixel just to add delay */
        w = h = 1;
        x = y = 0;
    }
    put_image(gif, w, h, x, y);
    gif->nframes++;
    if (gif->bgindex < 0) {
        tmp = gif->back;
        gif->back = gif->frame;
        gif->frame = tmp;
    }
}

void
ge_close_gif(ge_GIF* gif)
{
    write(gif->fd, ";", 1);
    close(gif->fd);
    free(gif);
}

```

BAB III **SCREENSHOT**

1. input :



```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/MAGA.jpeg
Pilih metode perhitungan error:
1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)
Masukkan nomor metode (1-5): 1
Masukkan nilai threshold: 50
Masukkan ukuran blok minimum: 50
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/MAGAHASIL.jpeg
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/MAGAGIF.jpeg
Memproses gambar...
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/MAGAHASIL.jpeg
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/MAGAGIF.jpeg
Waktu pemrosesan: 2289 ms
Ukuran Sebelum Kompresi: 107629 bytes
Ukuran Setelah Kompresi: 271140 bytes
Rasio Kompresi: -151.921%
Kedalaman Maksimum: 8
Banyak Simpul: 17637
```

output :



gif :



2. input :



```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/ishow_speed.jpeg  
Pilih metode perhitungan error:
```

1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)

```
Masukkan nomor metode (1-5): 2
```

```
Masukkan nilai threshold: 40
```

```
Masukkan ukuran blok minimum: 50
```

```
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0
```

```
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/ishow_speedHASIL.jpeg
```

```
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/ishow_speedGIF.gif
```

```
Memproses gambar...
```

```
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/ishow_speedHASIL.jpeg
```

```
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/ishow_speedGIF.gif
```

```
Waktu pemrosesan: 2503 ms
```

```
Ukuran Sebelum Kompresi: 277757 bytes
```

```
Ukuran Setelah Kompresi: 63255 bytes
```

```
Rasio Kompresi: 77.2265%
```

```
Kedalaman Maksimum: 8
```

```
Banyak Simpul: 369
```

output :



gif :



3. input :



Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/tariffs.jpeg
Pilih metode perhitungan error:

1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)

Masukkan nomor metode (1-5): 3

Masukkan nilai threshold: 60

Masukkan ukuran blok minimum: 70

Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0

Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/tariffsHASIL.jpeg

Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/tariffsGIF.gif

Memproses gambar...

Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/tariffsHASIL.jpeg

GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/tariffsGIF.gif

Waktu pemrosesan: 217 ms

Ukuran Sebelum Kompresi: 13052 bytes

Ukuran Setelah Kompresi: 59736 bytes

Rasio Kompresi: -357.677%

Kedalaman Maksimum: 6

Banyak Simpul: 1081

output :



gif :



4. input :

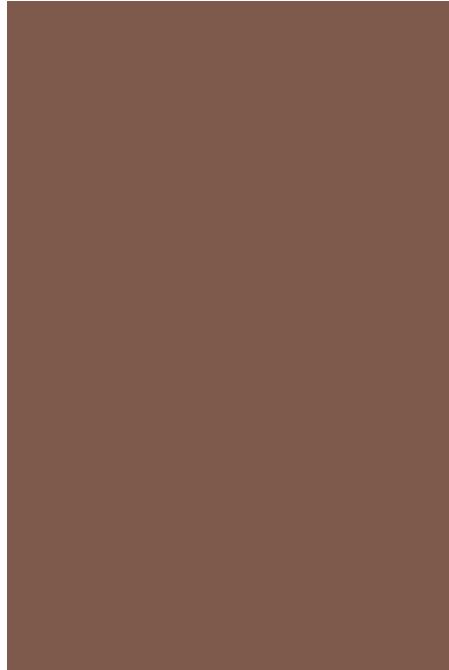


```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/image.jpeg
Pilih metode perhitungan error:
1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)
Masukkan nomor metode (1-5): 4
Masukkan nilai threshold: 4
Masukkan ukuran blok minimum: 40
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/image1.jpeg
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/image2.gif
Memproses gambar...
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/image1.jpeg
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/image2.gif
Waktu pemrosesan: 33629 ms
Ukuran Sebelum Kompresi: 1.87828e+06 bytes
Ukuran Setelah Kompresi: 2.74897e+06 bytes
Rasio Kompresi: -46.3558%
Kedalaman Maksimum: 10
Banyak Simpul: 163433
```

output :



gif :



5. input :



```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/bron.jpeg  
Pilih metode perhitungan error:
```

1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)

```
Masukkan nomor metode (1-5): 5
```

```
Masukkan nilai threshold: 0
```

```
Masukkan ukuran blok minimum: 60
```

```
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0
```

```
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/bronHASIL.jpeg
```

```
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/bronGIF.gif
```

```
Memproses gambar...
```

```
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/bronHASIL.jpeg
```

```
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/bronGIF.gif
```

```
Waktu pemrosesan: 1528 ms
```

```
Ukuran Sebelum Kompresi: 79829 bytes
```

```
Ukuran Setelah Kompresi: 307030 bytes
```

```
Rasio Kompresi: -284.61%
```

```
Kedalaman Maksimum: 8
```

```
Banyak Simpul: 21845
```

output :



gif :



6. input :



```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/lebron.jpeg  
Pilih metode perhitungan error:
```

1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)

```
Masukkan nomor metode (1-5): 4
```

```
Masukkan nilai threshold: 5
```

```
Masukkan ukuran blok minimum: 50
```

```
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 50
```

```
Rasio kompresi tidak valid. Silakan masukkan nilai antara 0.0 dan 1.0: 0.5
```

```
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/lebronHasil.jpeg
```

```
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/lebronGif.gif
```

```
Memproses gambar...
```

```
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/lebronHasil.jpeg
```

```
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/lebronGif.gif
```

```
Waktu pemrosesan: 8685 ms
```

```
Ukuran Sebelum Kompresi: 236406 bytes
```

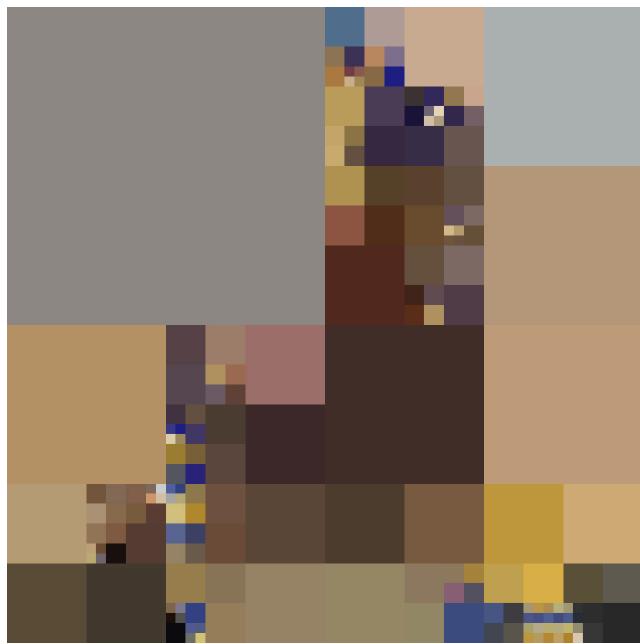
```
Ukuran Setelah Kompresi: 118817 bytes
```

```
Rasio Kompresi: 49.740277%
```

```
Kedalaman Maksimum: 7
```

```
Banyak Simpul: 241
```

output :



gif :



7. input :

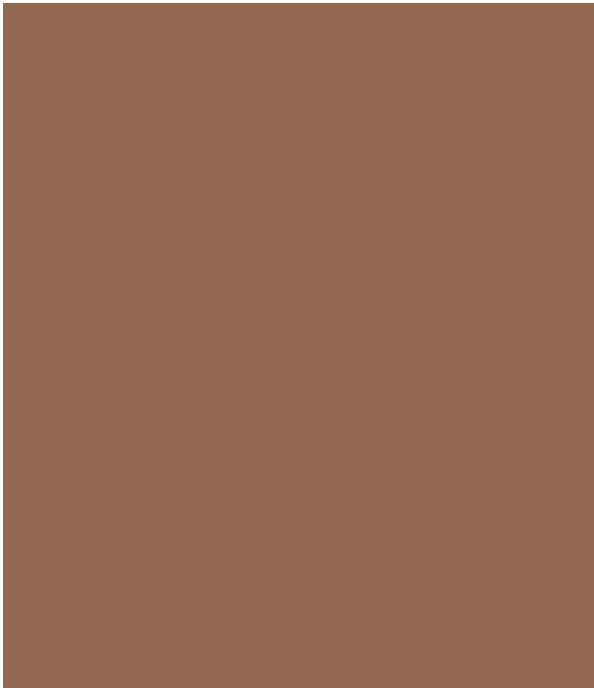


```
Masukkan alamat absolut ke gambar input (contoh: test/a.jpg): /home/owen/KULIAH/Tucil2 Stima/test/bronny.jpeg
Pilih metode perhitungan error:
1. Variansi
2. Rata-rata Deviasi Absolut (MAD)
3. Selisih Piksel Maksimum
4. Entropi
5. SSIM (Structural Similarity Index)
Masukkan nomor metode (1-5): 2
Masukkan nilai threshold: 40
Masukkan ukuran blok minimum: 70
Masukkan target rasio kompresi (0 untuk menonaktifkan mode ini, 1.0 untuk 100%): 0.4
Masukkan alamat absolut untuk menyimpan gambar hasil kompresi (contoh: test/b.png): /home/owen/KULIAH/Tucil2 Stima/test/bronnyHASIL.jpeg
Masukkan alamat absolut untuk menyimpan GIF (contoh: test/process.gif): /home/owen/KULIAH/Tucil2 Stima/test/bronnyGIF.gif
Memproses gambar...
Gambar berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/bronnyHASIL.jpeg
GIF berhasil disimpan di /home/owen/KULIAH/Tucil2 Stima/test/bronnyGIF.gif
Waktu pemrosesan: 9960 ms
Ukuran Sebelum Kompresi: 271358 bytes
Ukuran Setelah Kompresi: 158747 bytes
Rasio Kompresi: 41.499053%
Kedalaman Maksimum: 11
Banyak Simpul: 3889
```

output :



gif :



BAB IV ANALISIS

Analisis kompleksitas dari program utama:

Misal dilakukan kompresi terhadap image yang terdiri dari N baris dan M kolom. Alur rekursi dari *divide and conquer* dapat dibayangkan sebagai suatu pohon. Setiap simpul pada pohon tersebut memiliki suatu kedalaman relatif terhadap simpul akar (simpul akar adalah simpul pertama yang dipanggil pada proses *divide*). Pertama-tama, kelompokkan simpul-simpul pada pohon tersebut. Dua buah simpul berada di kelompok yang sama apabila kedua simpul tersebut

memiliki kedalaman yang sama. Perhatikan bahwa pada kasus terburuk akan terbentuk sebanyak $\log_4(NM)$ buah kelompok. Pada saat berpindah dari suatu simpul pada kedalaman d ke simpul-simpul anaknya pada kedalaman $d + 1$, simpul tersebut akan dipecah menjadi 4 anak terpisah. Jika pada kedalaman d terdapat sebanyak X buah simpul, pada kedalaman $d + 1$ terdapat sebanyak $4X$ buah simpul pada kasus terburuk. Selanjutnya, ditemukan bahwa pada kedalaman d akan terdapat sebanyak 4^d simpul (simpul akar dianggap memiliki kedalaman 0). Banyaknya simpul maksimal yang mungkin adalah sebanyak NM . Menghitung maksimum banyaknya kelompok ekuivalen dengan menghitung kedalaman maksimum dari suatu simpul. $4^d \leq NM \Rightarrow d \leq \log_4(NM)$. Terbukti bahwa pada kasus terburuk, ada sebanyak $\log_4(NM)$ buah kelompok. Terakhir untuk setiap kelompok, jelas bahwa setiap simpul mewakili sekelompok pixel yang terpisah dari simpul lainnya. Banyaknya pixel yang diproses oleh tiap kelompok ada sebanyak NM pixel. Akhirnya dapat disimpulkan bahwa kompleksitas waktu dari program ini sebesar $O(NM\log_4(NM))$.

BAB V

BONUS

1. SSIM Structural Similarity Index (SSIM) :

Fungsi CalculateSSIM merupakan implementasi dari perhitungan Structural Similarity Index (SSIM) antara sebuah blok gambar dan warna rata-ratanya (avgColor). SSIM digunakan untuk menilai kemiripan struktural antara dua gambar atau blok piksel, yang dalam konteks ini dipakai sebagai ukuran keseragaman warna blok pada proses kompresi Quadtree. Fungsi ini bekerja dengan mengakumulasi nilai statistik dari tiap kanal warna (R, G, B) secara terpisah: jumlah piksel (sum1), kuadrat piksel (sum1²), dan hasil kali silang antara piksel dan warna rata-rata (sum12). Dari sini dihitung rata-rata (mean), variansi, dan kovariansi untuk masing-masing kanal. SSIM kemudian dihitung berdasarkan rumus standar dengan konstanta stabilisasi C1 dan C2, dan nilai akhir SSIM diambil sebagai rata-rata dari SSIM kanal R, G, dan B. Fungsi ini membantu mengukur seberapa "seragam" blok piksel terhadap warna rata-ratanya—semakin tinggi nilai SSIM (mendekati 1), semakin seragam blok tersebut, yang berarti cocok untuk dikompresi sebagai satu simpul dalam Quadtree.

2. Target Persentase Kompresi :

Pada blok else ini program mengaktifkan mode pencarian otomatis untuk mencapai rasio kompresi yang diinginkan pengguna. Pertama, target Compression Ratio diubah ke skala persentase, lalu variabel L dan R diinisialisasi dengan batas bawah (low) dan atas (high) dari threshold yang valid. Dengan menggunakan binary search sebanyak 20 iterasi, pada setiap langkah dihitung titik tengah $M = (L + R)/2$, lalu quadtree dibangun ulang dengan threshold M dan blok terkecil 1×1. Gambar hasil rekonstruksi sementara disimpan, kemudian Calculate Compression Ratio memeriksa berapa persen ukuran file telah terkompresi. Jika rasio kompresi masih di bawah target, batas bawah L digeser ke M; jika sudah melebihi atau sama, batas atas R digeser ke M. Setelah 20 iterasi, nilai threshold akan mendekati titik optimal yang menghasilkan kompresi sesuai target tanpa perlu menebak secara manual. Alasan pemilihan 20 iterasi adalah karena pada tiap iterasi binary search, selisih jarak antara L dan R akan menjadi setengah dari jarak sebelumnya. 20 iterasi cukup untuk membuat L dan R konvergen ke suatu nilai.

3. GIF :

Gif dibangun berdasarkan Quadtree yang dibentuk pada tahap *divide*. Untuk dapat membentuk gif, perlu dibangun image untuk setiap kedalaman dimulai dari kedalaman 0. Algoritma paling tepat untuk mensimulasikan proses ini adalah BFS. BFS mengunjungi setiap simpul mulai dari simpul dengan kedalaman terkecil hingga simpul dengan kedalaman terbesar. Dengan urutan BFS, gif dapat dibentuk per kedalaman. Digunakan juga algoritma *neural quantizer* dalam proses ini karena file .gif memiliki batasan yaitu hanya dapat menampung 256 warna berbeda per *frame*. Algoritma *neural quantizer* bertujuan untuk melakukan kompresi warna sehingga total warna yang digunakan tidak melebihi 256 warna berbeda, namun warna-warna yang dipilih dapat dengan jelas menggambarkan gambar yang di-compress. Jika total warna berbeda yang ada pada suatu gambar tidak melebihi 256 warna, maka *neural quantizer* tidak digunakan.

LINK REPOSITORY

https://github.com/BP04/Tucil2_13523067_13523103

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	✓	

