Kompresi Gambar Dengan Metode Quadtree

Laporan Tugas Kecil 2

Disusun untuk memenuhi tugas kecil mata kuliah IF2211 Strategi Algoritma

oleh

Sebastian Hung Yansen 13523070

DAFTAR ISI

BAB I
DESKRIPSI MASALAH

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil. Pendekatan ini merupakan contoh dari strategi *divide and conquer*, yaitu memecah permasalahan besar menjadi bagian-bagian lebih kecil dan lebih sederhana, menyelesaikan tiap bagian secara terpisah, lalu menggabungkan hasilnya. Dalam konteks sebuah Quadtree, gambar dibagi menjadi empat bagian, dan pembagian ini diulang secara rekursif sampai diperoleh area-area yang cukup seragam. Quadtree sendiri direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam sedangkan simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Masing-masing simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas *pixel* dalam area tersebut. Struktur ini memungkinkan pengkodean data yang lebih efisien dengan menghilangkan redundansi pada area yang seragam.

Quadtree sering digunakan dalam pengolahan gambar termasuk kompresi gambar. Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas *pixel*. Proses kompresi gambar dimulai dengan membagi gambar menjadi empat bagian dan memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan sistem warna RGB (*Red, Green, Blue*) pada kumpulan *pixel* didalamnya. Apabila tidak seragam, bagian tersebut akan dibagi lagi hingga mencapai tingkat keseragaman tertentu.

Tugas yang diberikan adalah membuat aplikasi kompresi gambar yang menggunakan quadtree. Pengguna dapat mengubah parameter-parameter seperti metode perhitungan variansi, threshold variansi, dan minimum block size.

BAB II
PENYELESAIAN

Berikut adalah langkah-langkah penyelesaiannya beserta penjelasan algoritma *divide and conquer*:

1. Program akan bertanya kepada pengguna terkait alamat gambar yang ingin dikompres, metode pengukuran error, ambang batas, ukuran minimum blok, dan alamat gambar hasil kompresi.
2. Program akan memuat gambar serta membuat *quadtree* berdasarkan ambang batas, gambar, dan ukuran minimum blok.
3. Program membagi gambar menjadi blok-blok yang lebih kecil secara rekursif dalam bentuk kuadran apabila *error* melebihi ambang batas atau ukuran blok melebihi ukuran minimum blok. Apabila angka *error* masih di bawah ambang batas atau blok terlalu kecil, blok tersebut tidak dibagi dan dijadikan *leaf* dengan warna rata-rata. Hasil dari setiap kuadran digabungkan menjadi *quadtree* dengan masing-masing node mewakili blok dan anak-anaknya mewakili kuadran. Masing-masing blok diproses secara independen.
4. Setelah itu, program menelusuri *quadtree* dan membentuk ulang gambar dengan *leaf nodes* untuk mengisi blok dengan warna rata-ratanya
5. Setelah itu gambar disimpan dan program mengeluarkan hasil waktu eksekusi, ukuran gambar awal, ukuran gambar akhir, persentase kompresi, kedalaman pohon, dan banyak simpul pada pohon.

Program akan otomatis keluar apabila ada input yang tidak sesuai atau proses gagal karena input tidak valid atau terjadi error saat eksekusi.

BAB III
IMPLEMENTASI

3.1. Fungsi dan Prosedur

Fungsi/Prosedur	Tujuan

`double ErrorMeasurementMethod::computeVariance(const vector<vector<vector<int>>>& block) {`
`double ErrorMeasurementMethod::computeMAD(const vector<vector<vector<int>>>& block) {`
`double ErrorMeasurementMethod::computeMaxPixelDiff(const vector<vector<vector<int>>>& block) {`
`double ErrorMeasurementMethod::computeEntropy(const vector<vector<vector<int>>>& block) {`
	Metode pengukuran *error* yang dapat dipilih pengguna

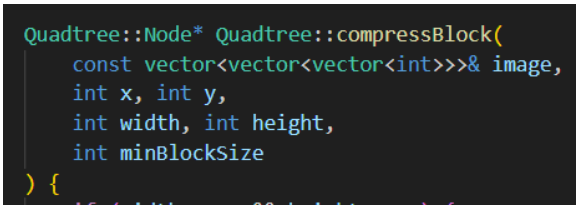
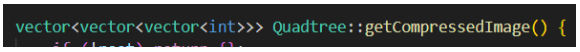
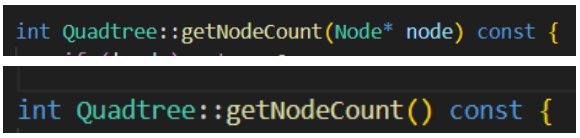
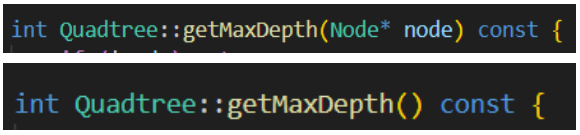
image.cpp	

`Image::Image() : width(0), height(0), channels(0) {}`
 	
`Image::~Image() {`
`  pixels.clear();`
`}`
	Constructor, Destructor Gambar

`long long Image::getFileSize(const string& filePath) const {`	Mengambil data ukuran gambar
`bool Image::loadImage(const string& filepath) {`	Memuat gambar yang akan diproses
`bool Image::saveImage(const string& filepath) const {`	Menyimpan gambar yang sudah diproses
`vector<vector<vector<int>>> Image::getPixelMatrix() const {` `void Image::setPixelMatrix(const vector<vector<vector<int>>>& newPixels) {`	Pixel Matrix untuk mengubah representasi pixel dalam gambar menjadi matrix agar lebih mudah untuk diolah
quadtree.cpp	

`Quadtree::Node::Node(int x, int y, int size, bool isLeaf, vector<int> color)`
`  : x(x), y(y), size(size), isLeaf(isLeaf), color(color) {`
`  for (int i = 0; i < 4; i++) {`
`  children[i] = nullptr;`
`  }`
`}`
 	
`Quadtree::Node::~Node() {`
`  for (int i = 0; i < 4; i++) {`
`  delete children[i];`
`  }`
`}`
 	
`Quadtree::Quadtree(int threshold, ErrorMethod method) {`
`  this->threshold = threshold;`
`  this->method = method;`
`}`
 	
`Quadtree::~Quadtree() {`
`}`
	Constructor, Destructor node dan quadtree

`void Quadtree::build(const vector<vector<vector<int>>>& image, int imgWidth, int imgHeight, int minBlockSize) {`	Membangun nodetree

	Fungsi membagi dan kompres blok. Pemanggilan fungsi dilakukan secara rekursif untuk masing-masing kuadran
	Fungsi mengambil data gambar yang sudah diproses
	Fungsi membangun ulang gambar berdasarkan matrix
	Fungsi mendapatkan data jumlah node
	Fungsi mendapatkan kedalaman pohon

3.2. Source Code Program

1. File main.cpp

src > main.cpp > main()

```cpp
#include <iostream>
#include <string>
#include <chrono>
#include "include\quadtree.hpp"
#include "include\errorMeasurements.hpp"
using namespace std;

int main() {
    string inputAddress; // Alamat gambar yang akan dikompresi
    string outputAddress; // Alamat gambar hasil kompresi
    int errorMeasurementMethod; // Metode kompresi yang dipilih
    int threshold; // Ambang batas untuk kompresi
    int minimumBlockSize;
    bool isSaved; // Menyimpan status penyimpanan gambar

    cout << "Masukkan alamat gambar: ";
    getline(cin, inputAddress);
    Image img;
    if (!img.loadImage(inputAddress)) {
        cout << "Gagal memuat gambar." << endl;
        return 1;
    }

    do {
        cout << "Pilih metode pengukuran error:" << endl;
        cout << "1. Variance" << endl;
        cout << "2. Mean Absolute Deviation (MAD)" << endl;
        cout << "3. Max Pixel Difference" << endl;
        cout << "4. Entropy" << endl;
        cout << "Masukkan pilihan (1-4): ";
        cin >> errorMeasurementMethod;

        if (cin.fail() || errorMeasurementMethod < 1 || errorMeasurementMethod > 4) {
            cout << "Input tidak valid. Masukkan angka antara 1 dan 4." << endl;
            cin.clear();
            cin.ignore(numeric_limits<streamsize>::max(), '\n');
        }
```

```
38      } while (errorMeasurementMethod < 1 || errorMeasurementMethod > 4);
39
40      do {
41          cout << "Masukkan ambang batas (threshold, angka positif): ";
42          cin >> threshold;
43
44          if (cin.fail() || threshold <= 0) {
45              cout << "Input tidak valid. Masukkan angka positif." << endl;
46              cin.clear();
47              cin.ignore(numeric_limits<streamsize>::max(), '\n');
48          }
49      } while (threshold <= 0);
50
51      do {
52          cout << "Masukkan ukuran minimum blok (angka positif): ";
53          cin >> minimumBlockSize;
54
55          if (cin.fail() || minimumBlockSize <= 0) {
56              cout << "Input tidak valid. Masukkan angka positif." << endl;
57              cin.clear();
58              cin.ignore(numeric_limits<streamsize>::max(), '\n');
59          }
60      } while (minimumBlockSize <= 0);
61
62      if (minimumBlockSize < 1) {
63          cout << "Ukuran minimum blok tidak valid" << endl;
64          return 1;
65      }
66
67      cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
68      cout << "Masukkan alamat gambar hasil kompresi: ";
69      getline(cin, outputAddress);
```

```
    cin.ignore(std::numeric_limits<std::streamsize>::max(), '\n');
    cout << "Masukkan alamat gambar hasil kompresi: ";
    getline(cin, outputAddress);

    Quadtree qt(threshold, static_cast<ErrorMethod>(errorMeasurementMethod-1));
    std::vector<std::vector<std::vector<int>>> pixelMatrix = img.getPixelMatrix();
    std::vector<std::vector<std::vector<int>>> compressedMatrix;
    auto startTime = chrono::high_resolution_clock::now();

    if (errorMeasurementMethod == 1) {
        cout << "Menggunakan metode Variance" << endl;
        qt.build(pixelMatrix, img.getWidth(), img.getHeight(), minimumBlockSize);
        compressedMatrix = qt.getCompressedImage();
    } else if (errorMeasurementMethod == 2) {
        cout << "Menggunakan metode Mean Absolute Deviation (MAD)" << endl;
        qt.build(pixelMatrix, img.getWidth(), img.getHeight(), minimumBlockSize);
        compressedMatrix = qt.getCompressedImage();
    } else if (errorMeasurementMethod == 3) {
        cout << "Menggunakan metode Max Pixel Difference" << endl;
        qt.build(pixelMatrix, img.getWidth(), img.getHeight(), minimumBlockSize);
        compressedMatrix = qt.getCompressedImage();
    } else if (errorMeasurementMethod == 4) {
        cout << "Menggunakan metode Entropy" << endl;
        qt.build(pixelMatrix, img.getWidth(), img.getHeight(), minimumBlockSize);
        compressedMatrix = qt.getCompressedImage();
    } else {
        cout << "Metode tidak valid" << endl;
        return 1;
    }
```

```
    img.setPixelMatrix(compressedMatrix);
    if (img.saveImage(outputAddress)) {
        isSaved = true;
    } else {
        isSaved = false;
    }

    auto endTime = chrono::high_resolution_clock::now();
    chrono::duration<double> executionTime = endTime - startTime;

    long long originalSize = img.getFileSize(inputAddress) / 1000;
    long long compressedSize = img.getFileSize(outputAddress) / 1000;
    double compressionPercentage = 100.0 * (1.0 - static_cast<double>(compressedSize) / originalSize);

    cout << "============================" << endl;
    cout << "Waktu eksekusi: " << executionTime.count() << " detik" << endl;
    cout << "Ukuran gambar sebelum: " << originalSize << " KB" << endl;
    cout << "Ukuran gambar setelah: " << compressedSize << " KB" << endl;
    cout << "Persentase kompresi: " << compressionPercentage << "%" << endl;
    cout << "Kedalaman pohon: " << qt.getMaxDepth() << endl;
    cout << "Banyak simpul pada pohon: " << qt.getNodeCount() << endl;
    if (isSaved) {
        cout << "Gambar berhasil dikompresi dan disimpan di " << outputAddress << endl;
    } else {
        cout << "Gambar gagal disimpan " << endl;
    }
    cout << "============================" << endl;

    return 0;
}
```

2. File errorMeasurements.cpp

```cpp
#include "include\errorMeasurements.hpp"
#include <cmath>
#include <algorithm>
#include <unordered_map>
using namespace std;

double ErrorMeasurementMethod::computeVariance(const vector<vector<vector<int>>>& block) {
    double varianceR = 0.0, varianceG = 0.0, varianceB = 0.0;
    int totalPixels = block.size() * block[0].size();

    double meanR = 0.0, meanG = 0.0, meanB = 0.0;
    for (const auto& row : block) {
        for (const auto& pixel : row) {
            meanR += pixel[0];
            meanG += pixel[1];
            meanB += pixel[2];
        }
    }
    meanR /= totalPixels;
    meanG /= totalPixels;
    meanB /= totalPixels;

    for (const auto& row : block) {
        for (const auto& pixel : row) {
            varianceR += (pixel[0] - meanR) * (pixel[0] - meanR);
            varianceG += (pixel[1] - meanG) * (pixel[1] - meanG);
            varianceB += (pixel[2] - meanB) * (pixel[2] - meanB);
        }
    }
    varianceR /= totalPixels;
    varianceG /= totalPixels;
    varianceB /= totalPixels;

    return (varianceR + varianceG + varianceB) / 3.0;
}

```

```cpp
double ErrorMeasurementMethod::computeMAD(const vector<vector<vector<int>>>& block) {
    double MADR = 0.0, MADG = 0.0, MADB = 0.0;
    int totalPixels = block.size() * block[0].size();

    double meanR = 0, meanG = 0, meanB = 0;
    for (const auto& row : block) {
        for (const auto& pixel : row) {
            meanR += pixel[0];
            meanG += pixel[1];
            meanB += pixel[2];
        }
    }
    meanR /= totalPixels;
    meanG /= totalPixels;
    meanB /= totalPixels;

    for (const auto& row : block) {
        for (const auto& pixel : row) {
            MADR += abs(pixel[0] - meanR);
            MADG += abs(pixel[1] - meanG);
            MADB += abs(pixel[2] - meanB);
        }
    }
    MADR /= totalPixels;
    MADG /= totalPixels;
    MADB /= totalPixels;

    return (MADR + MADG + MADB) / 3.0;
}
```

```cpp
double ErrorMeasurementMethod::computeMaxPixelDiff(const vector<vector<vector<int>>>& block) {
    double maxR = 0.0, maxG = 0.0, maxB = 0.0, minR = 255.0, minG = 255.0, minB = 255.0;
    int totalPixels = block.size() * block[0].size();

    for (const auto& row : block) {
        for (const auto& pixel : row) {
            if (pixel[0] > maxR) maxR = pixel[0];
            if (pixel[0] < minR) minR = pixel[0];
            if (pixel[1] > maxG) maxG = pixel[1];
            if (pixel[1] < minG) minG = pixel[1];
            if (pixel[2] > maxB) maxB = pixel[2];
            if (pixel[2] < minB) minB = pixel[2];
        }
    }

    return ((maxR - minR) + (maxG - minG) + (maxB - minB)) / 3.0;
}
```

```cpp
double ErrorMeasurementMethod::computeEntropy(const vector<vector<vector<int>>>& block) {
    unordered_map<int, int> entR, entG, entB;
    int totalPixels = block.size() * block[0].size();

    for (const auto& row : block) {
        for (const auto& pixel : row) {
            entR[pixel[0]]++;
            entG[pixel[1]]++;
            entB[pixel[2]]++;
        }
    }

    auto calculateEntropy = [&](const unordered_map<int, int>& freqMap) -> double {
        double entropy = 0.0;
        for (const auto& [value, count] : freqMap) {
            double probability = static_cast<double>(count) / totalPixels;
            if (probability > 0) {
                entropy -= probability * log2(probability);
            }
        }
        return entropy;
    };

    double entropyR = calculateEntropy(entR);
    double entropyG = calculateEntropy(entG);
    double entropyB = calculateEntropy(entB);

    return (entropyR + entropyG + entropyB) / 3.0;
}
```

3. File image.cpp

```cpp
#define STB_IMAGE_IMPLEMENTATION
#define STB_IMAGE_WRITE_IMPLEMENTATION
#define STBI_NO_EXIF
#include "include/stb_image.h"
#include "include/stb_image_write.h"
#include "include/image.hpp"
#include <iostream>
#include <sstream>
#include <filesystem>
#include <fstream>
using namespace std;

Image::Image() : width(0), height(0), channels(0) {}

Image::~Image() {
    pixels.clear();
}

long long Image::getFileSize(const string& filePath) const {
    ifstream file(filePath, ios::binary | ios::ate);
    if (!file.is_open()) {
        cerr << "Error: File gagal dibuka: " << filePath << endl;
        return -1;
    }
    return file.tellg();
}

bool Image::loadImage(const string& filepath) {
    unsigned char* data = stbi_load(filepath.c_str(), &width, &height, &channels, 3);
    if (!data) {
        cerr << "Error: Gambar gagal di-load: " << filepath << endl;
        return false;
    }

    channels = 3;

```

```
    rotated = false;

    if (height > width) { // If image is portrait
        vector<unsigned char> rotatedPixels(height * width * channels);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int srcIdx = (y * width + x) * channels;
                int dstIdx = (x * height + (height - y - 1)) * channels;

                for (int c = 0; c < channels; c++) {
                    rotatedPixels[dstIdx + c] = data[srcIdx + c];
                }
            }
        }

        swap(width, height);
        pixels = move(rotatedPixels);
        rotated = true;
    } else {
        pixels.assign(data, data + (width * height * channels));
    }

    stbi_image_free(data);
    return true;
}

bool Image::saveImage(const string& filepath) const {
    if (pixels.empty()) {
        cerr << "Error: Data gambar tidak ada" << endl;
        return false;
    }

    std::filesystem::path filePathObj(filepath);
    std::filesystem::path directory = filePathObj.parent_path();
```

```
    if (!directory.empty() && !std::filesystem::exists(directory)) {
        if (!std::filesystem::create_directories(directory)) {
            return false;
        }
    }

    int outWidth = width, outHeight = height;
    vector<unsigned char> outputPixels = pixels;

    if (rotated) { // Rotate back
        vector<unsigned char> rotatedPixels(width * height * channels);

        for (int y = 0; y < height; y++) {
            for (int x = 0; x < width; x++) {
                int srcIdx = (y * width + x) * channels;
                int dstIdx = ((width - x - 1) * height + y) * channels;

                for (int c = 0; c < channels; c++) {
                    rotatedPixels[dstIdx + c] = pixels[srcIdx + c];
                }
            }
        }

        outputPixels = move(rotatedPixels);
        swap(outWidth, outHeight);
    }

    if (!stbi_write_png(filepath.c_str(), outWidth, outHeight, channels, outputPixels.data(), outWidth * channels)) {
        cerr << "Error: Gagal menyimpan gambar " << filepath << endl;
        return false;
    }

    return true;
}
```

```
vector<vector<vector<int>>> Image::getPixelMatrix() const {
    vector<vector<vector<int>>> matrix(height, vector<vector<int>>(width, vector<int>(channels)));

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int index = (y * width + x) * channels;
            for (int c = 0; c < channels; c++) {
                matrix[y][x][c] = static_cast<int>(pixels[index + c]);
            }
        }
    }

    return matrix;
}

void Image::setPixelMatrix(const vector<vector<vector<int>>>& newPixels) {
    this->height = newPixels.size();
    this->width = (this->height > 0) ? newPixels[0].size() : 0;

    if (this->width > 0 && this->height > 0) {
        int newChannels = newPixels[0][0].size();
        if (newPixels.empty() || newPixels[0].empty()) return;
        channels = newPixels[0][0].size();
    }

    pixels.clear();
    pixels.reserve(this->height * this->width * channels);

    for (const auto& row : newPixels) {
        for (const auto& pixel : row) {
            for (int c = 0; c < channels; c++) {
                pixels.push_back(static_cast<unsigned char>(pixel[c]));
            }
        }
    }
}
```

4. File quadtree.cpp

src > quadtree.cpp > ...

```cpp
#include "include\quadtree.hpp"
#include <iostream>
#include <queue>
using namespace std;


Quadtree::Node::Node(int x, int y, int size, bool isLeaf, vector<int> color)
    : x(x), y(y), size(size), isLeaf(isLeaf), color(color) {
    for (int i = 0; i < 4; i++) {
        children[i] = nullptr;
    }
}

Quadtree::Node::~Node() {
    for (int i = 0; i < 4; i++) {
        delete children[i];
    }
}

Quadtree::Quadtree(int threshold, ErrorMethod method) {
    this->threshold = threshold;
    this->method = method;
}

Quadtree::~Quadtree() {
}

void Quadtree::build(const vector<vector<vector<int>>>& image, int imgWidth, int imgHeight, int minBlockSize) {
    originalWidth = imgWidth;
    originalHeight = imgHeight;
    root = compressBlock(image, 0, 0, originalWidth, originalHeight, minBlockSize);
}

```

```
Quadtree::Node* Quadtree::compressBlock(
    const vector<vector<vector<int>>>& image,
    int x, int y,
    int width, int height,
    int minBlockSize
) {
    if (width == 1 && height == 1) {
        return new Node(x, y, width, false, vector<int>{0, 0, 0});
    }

    vector<vector<vector<int>>> block(height, vector<vector<int>>(width, vector<int>(3)));
    for (int i = 0; i < height; i++)
        for (int j = 0; j < width; j++)
            block[i][j] = image[x + i][y + j]; //nilai RGB

    double error = 0;

    switch (this->method) {
        case ErrorMethod::VARIANCE:
            error = ErrorMeasurementMethod::computeVariance(block);
            break;
        case ErrorMethod::MEAN_ABSOLUTE_DEVIATION:
            error = ErrorMeasurementMethod::computeMAD(block);
            break;
        case ErrorMethod::MAX_PIXEL_DIFFERENCE:
            error = ErrorMeasurementMethod::computeMaxPixelDiff(block);
            break;
        case ErrorMethod::ENTROPY:
            error = ErrorMeasurementMethod::computeEntropy(block);
            break;
    }

    if (error <= threshold || width <= minBlockSize || height <= minBlockSize) {
        vector<int> avgColor(3, 0);
        for (const auto& row : block)
            for (const auto& pixel : row)
```

```
            for (int c = 0; c < 3; c++)
                avgColor[c] += pixel[c];

    for (int c = 0; c < 3; c++)
        avgColor[c] /= (width * height);

    return new Node(x, y, width, true, avgColor);
}

int newWidth = width / 2;
int newHeight = height / 2;

vector<int> avgColor(3, 0);
if (x == 0 && y == 0 && width == image.size() && height == image[0].size()) {
    for (const auto& row : image) {
        for (const auto& pixel : row) {
            for (int c = 0; c < 3; c++) {
                avgColor[c] += pixel[c];
            }
        }
    }
    for (int c = 0; c < 3; c++) {
        avgColor[c] /= (width * height);
    }
}

Node* node = new Node(x, y, width, false, avgColor);

node->children[0] = compressBlock(image, x, y, newWidth, newHeight, minBlockSize);
node->children[1] = (width > newWidth) ? compressBlock(image, x, y + newWidth, width - newWidth, newHeight, minBlockSize) : nullptr;
node->children[2] = (height > newHeight) ? compressBlock(image, x + newHeight, y, newWidth, height - newHeight, minBlockSize) : nullptr;
node->children[3] = (width > newWidth && height > newHeight) ? compressBlock(image, x + newHeight, y + newWidth, width - newWidth, height - newHeight, minBlockSize) : nul
```

```
    return node;
}

vector<vector<vector<int>>> Quadtree::getCompressedImage() {
    if (!root) return {};

    vector<vector<vector<int>>> compressedImage(
        originalHeight, vector<vector<int>>(originalWidth, vector<int>(3, 255))
    );

    int frameNumber = 0;
    Image img;

    int maxDepth = getMaxDepth(root);
    for (int depth = 0; depth <= maxDepth; depth++) {
        reconstructImage(root, compressedImage, frameNumber, img, 0, depth);
    }

    return compressedImage;
}
```

```
void Quadtree::reconstructImage(Node* root, vector<vector<vector<int>>>& image, int& frameNumber, Image& img, int depth, int maxDepth) {
    if (!root) return;

    queue<Node*> q;
    q.push(root);

    while (!q.empty()) {
        int levelSize = q.size();
        for (int i = 0; i < levelSize; i++) {
            Node* node = q.front();
            q.pop();

            if (node->isLeaf) {
                for (int i = 0; i < node->size; i++) {
                    for (int j = 0; j < node->size; j++) {
                        int px = node->x + i;
                        int py = node->y + j;
                        if (px >= image.size() || py >= image[0].size()) continue;
                        image[px][py] = node->color;
                    }
                }
            } else {
                for (int i = 0; i < 4; i++) {
                    if (node->children[i]) q.push(node->children[i]);
                }
            }
        }
    }
}
```

```cpp
int Quadtree::getNodeCount(Node* node) const {
    if (!node) return 0;
    int count = 1;
    for (int i = 0; i < 4; i++) {
        count += getNodeCount(node->children[i]);
    }
    return count;
}

int Quadtree::getNodeCount() const {
    return getNodeCount(root);
}

int Quadtree::getMaxDepth(Node* node) const {
    if (!node) return 0;
    if (node->isLeaf) return 0;

    int maxChildDepth = 0;
    for (int i = 0; i < 4; i++) {
        if (node->children[i]) {
            maxChildDepth = max(maxChildDepth, getMaxDepth(node->children[i]));
        }
    }
    return maxChildDepth + 1;
}

int Quadtree::getMaxDepth() const {
    return getMaxDepth(root);
}
```

BAB IV
PENGUJIAN

<table>
<tr><th>No.</th><th>Input</th><th>Output</th></tr>
<tr><td rowspan="2">1</td><td>

```
Masukkan alamat gambar: ../test/ramen.jpg
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 1
Masukkan ambang batas: 7
Masukkan ukuran minimum blok: 5
Masukkan alamat gambar hasil kompresi: ../test/results/ramenCompressed.jpg
```

</td><td>

```
Menggunakan metode Variance
============================
Waktu eksekusi: 67.7242 detik
Ukuran gambar sebelum: 3238 KB
Ukuran gambar setelah: 2545 KB
Persentase kompresi: 21.4021%
Kedalaman pohon: 10
Banyak simpul pada pohon: 786473
Gambar berhasil dikompresi dan disimpan di ../test/results/ramenCompressed.jpg
============================
```

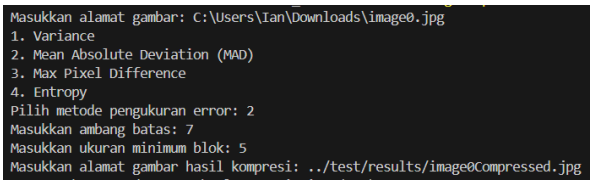
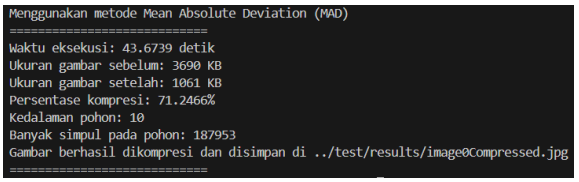
</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>2</td><td>

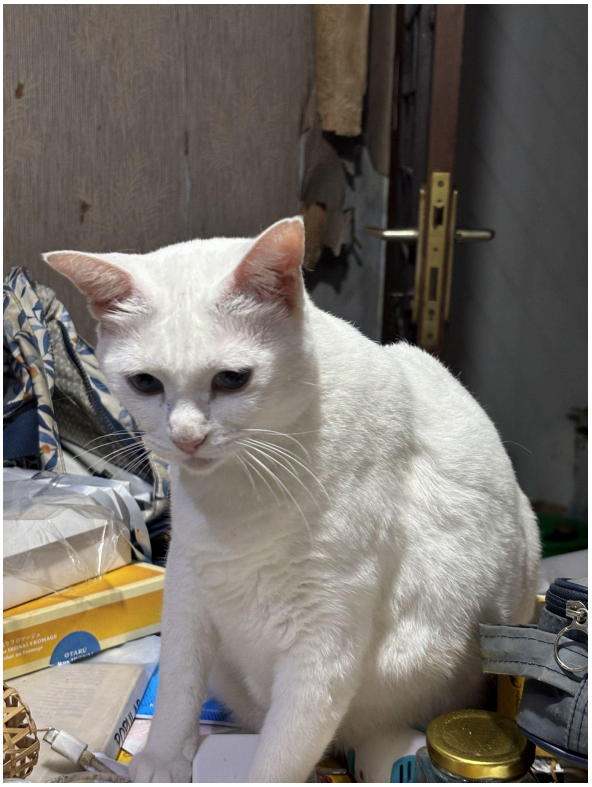
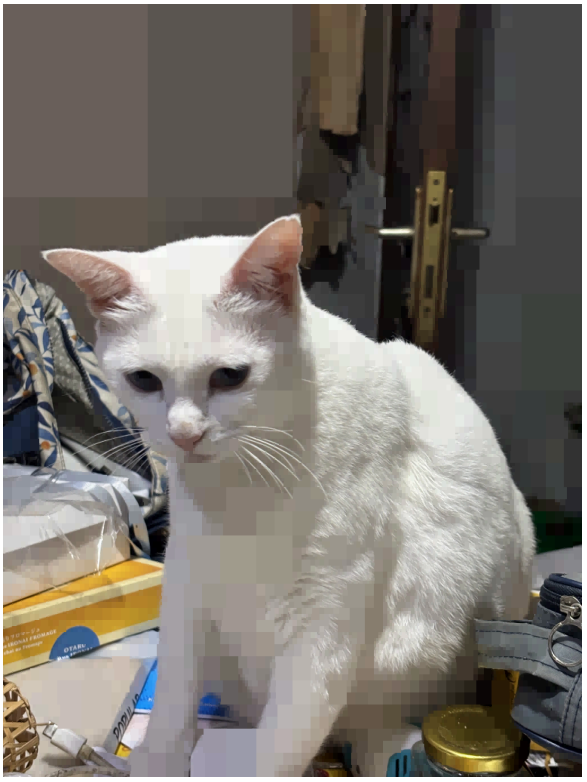
```
Masukkan alamat gambar: C:\Users\Ian\Pictures\mako.jpg
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 1
Masukkan ambang batas: 100
Masukkan ukuran minimum blok: 16
Masukkan alamat gambar hasil kompresi: ../test/results/makoCompressed.jpg
```

</td><td>

```
Menggunakan metode Variance
============================
Waktu eksekusi: 6.89127 detik
Ukuran gambar sebelum: 627 KB
Ukuran gambar setelah: 91 KB
Persentase kompresi: 85.4864%
Kedalaman pohon: 7
Banyak simpul pada pohon: 7873
Gambar berhasil dikompresi dan disimpan di ../test/results/makoCompressed.jpg
============================
```

</td></tr>
</table>

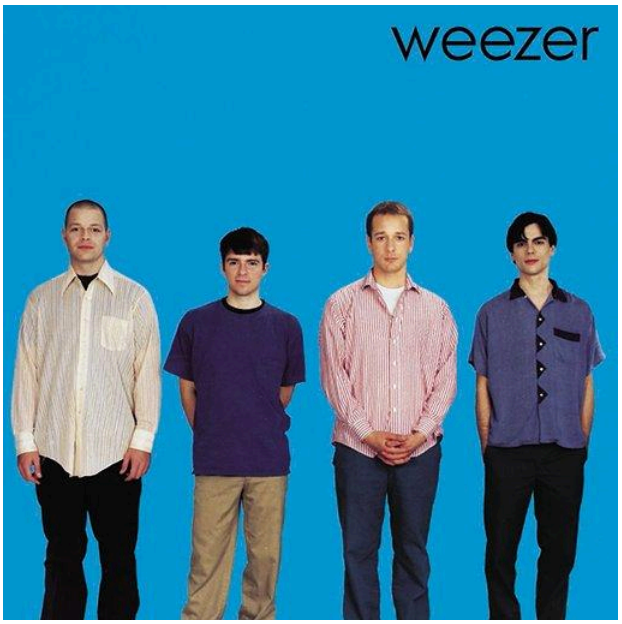
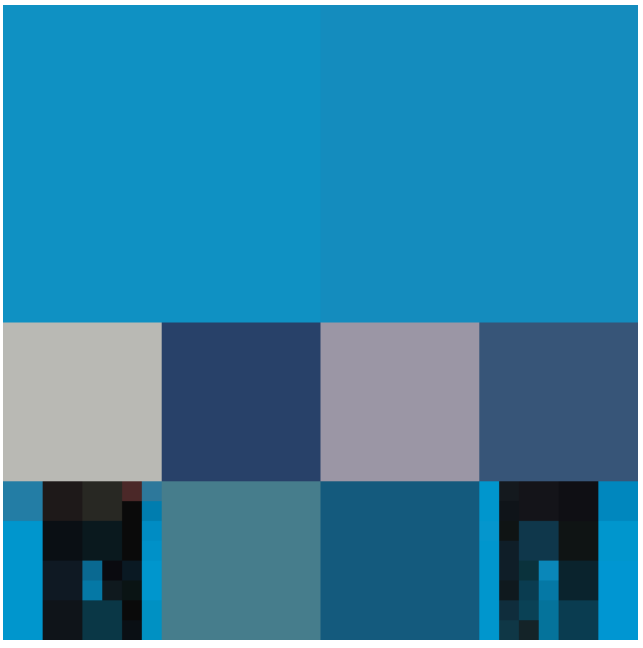
		
3	 Masukkan alamat gambar: C:\Users\Ian\Downloads\image0.jpg 1. Variance 2. Mean Absolute Deviation (MAD) 3. Max Pixel Difference 4. Entropy Pilih metode pengukuran error: 2 Masukkan ambang batas: 7 Masukkan ukuran minimum blok: 5 Masukkan alamat gambar hasil kompresi: ../test/results/image0Compressed.jpg	 Menggunakan metode Mean Absolute Deviation (MAD) ========== Waktu eksekusi: 43.6739 detik Ukuran gambar sebelum: 3690 KB Ukuran gambar setelah: 1061 KB Persentase kompresi: 71.2466% Kedalaman pohon: 10 Banyak simpul pada pohon: 187953 Gambar berhasil dikompresi dan disimpan di ../test/results/image0Compressed.jpg ==========

<table>
<tr>
<td></td>
<td>  </td>
<td>  </td>
</tr>
<tr>
<td rowspan="2">4</td>
<td>

```
Masukkan alamat gambar: ../test/Buddy Holly.jpg
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 2
Masukkan ambang batas: 50
Masukkan ukuran minimum blok: 8
Masukkan alamat gambar hasil kompresi: ../test/results/weezerCompressed.jpg
```

</td>
<td>

```
Menggunakan metode Mean Absolute Deviation (MAD)
==========
Waktu eksekusi: 0.467119 detik
Ukuran gambar sebelum: 39 KB
Ukuran gambar setelah: 8 KB
Persentase kompresi: 79.4872%
Kedalaman pohon: 5
Banyak simpul pada pohon: 97
Gambar berhasil dikompresi dan disimpan di ../test/results/weezerCompressed.jpg
==========
```

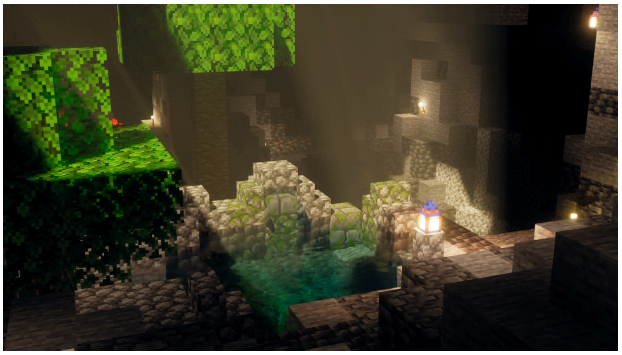
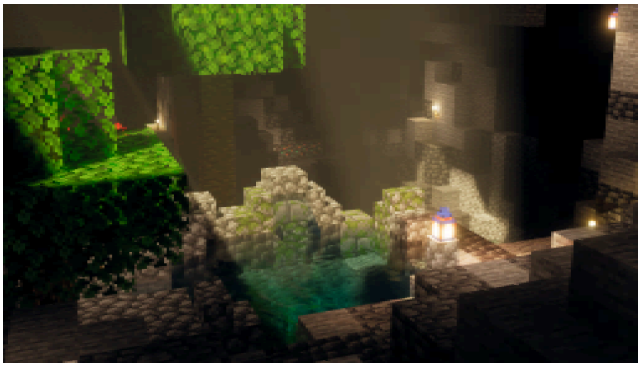
</td>
</tr>
<tr>
<td> 
</td>
<td>  </td>
</tr>
</table>

<table>
<tr><td rowspan="2">5</td><td>

```
Masukkan alamat gambar: ../test/minecraft.png
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 3
Masukkan ambang batas: 7
Masukkan ukuran minimum blok: 5
Masukkan alamat gambar hasil kompresi: ../test/results/minecraft Compressed.png
```

</td><td>

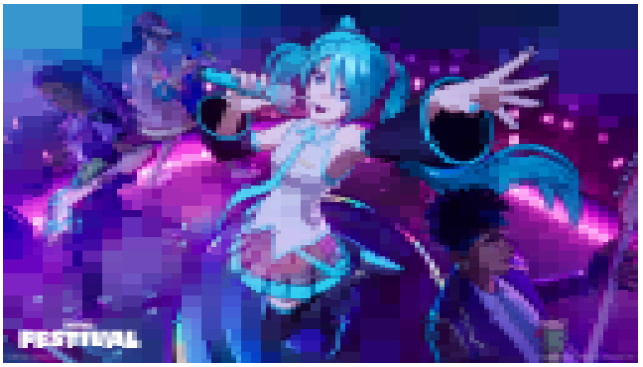
```
Menggunakan metode Max Pixel Difference
==========
Waktu eksekusi: 7.9359 detik
Ukuran gambar sebelum: 3849 KB
Ukuran gambar setelah: 249 KB
Persentase kompresi: 93.5308%
Kedalaman pohon: 8
Banyak simpul pada pohon: 64341
Gambar berhasil dikompresi dan disimpan di ../test/results/minecraft Compressed.png
==========
```

</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td rowspan="2">6</td><td>

```
Masukkan alamat gambar: ../test/mikuFest.jpg
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 3
Masukkan ambang batas: 100
Masukkan ukuran minimum blok: 16
Masukkan alamat gambar hasil kompresi: ../test/results/mikuCompressed.jpg
```

</td><td>

```
Menggunakan metode Max Pixel Difference
==========
Waktu eksekusi: 6.43753 detik
Ukuran gambar sebelum: 238 KB
Ukuran gambar setelah: 101 KB
Persentase kompresi: 57.563%
Kedalaman pohon: 7
Banyak simpul pada pohon: 9121
Gambar berhasil dikompresi dan disimpan di ../test/results/mikuCompressed.jpg
==========
```

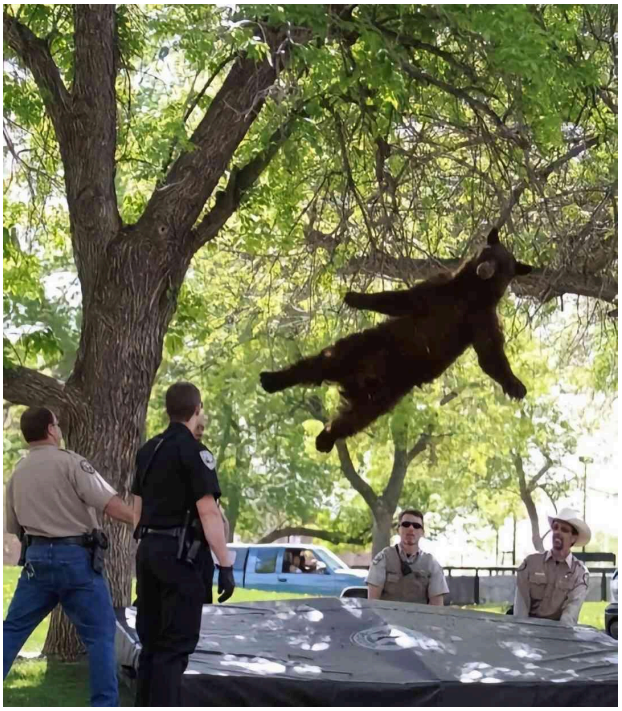
</td></tr>
<tr><td>  </td><td>  </td></tr>
<tr><td>7</td><td>

```
Masukkan alamat gambar: ../test/bear.jpg
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode pengukuran error: 4
Masukkan ambang batas: 7
Masukkan ukuran minimum blok: 5
Masukkan alamat gambar hasil kompresi: ../test/results/bearCompressed.jpg
```

</td><td>

```
Menggunakan metode Entropy
==========
Waktu eksekusi: 4.16341 detik
Ukuran gambar sebelum: 124 KB
Ukuran gambar setelah: 49 KB
Persentase kompresi: 60.4839%
Kedalaman pohon: 7
Banyak simpul pada pohon: 1849
Gambar berhasil dikompresi dan disimpan di ../test/results/bearCompressed.jpg
==========
```

</td></tr>
</table>

BAB V
KESIMPULAN

Berdasarkan implementasi kompresi gambar dengan metode quadtree, dapat disimpulkan bahwa penggunaan struktur quadtree dapat digunakan untuk kompresi gambar. Pada metode-metode tertentu, memasukkan nilai ambang batas yang terlalu besar dapat menyebabkan gambar menjadi 1 blok warna yang sangat besar dan tidak menyerupai gambar sedikitpun. Selain itu, ambang batas yang tinggi pada beberapa metode menyebabkan peletakkan blok yang tidak seragam dan ketidakseragaman tersebut sangat mudah terlihat.

Meskipun begitu, penggunaan algoritma *divide and conquer* pada quadtree menjadi sesuatu yang unik dan menarik untuk diamati terutama ketika membandingkan gambar awal dengan hasil akhir kompresi. Hasil akhir kompresi gambar menjadi file yang ukurannya lebih kecil dan gambar yang lebih kabur atau *pixelated* namun tidak kehilangan informasi penting pada level visual tertentu

LAMPIRAN

1. Repository

   https://github.com/SebastianHYH/Tucil2_13523070

2. Tabel Pemeriksaan

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	✓	