

Laporan Tugas Kecil 2

IF2211 Strategi Algoritma



Disusun oleh
Hasri Fayadh Muqaffa - 13523156

SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
SEMESTER 2 TAHUN 2024/2025

DAFTAR ISI

DAFTAR ISI.....	1
BAB 1.....	2
DESKRIPSI MASALAH.....	2
Spesifikasi Tugas Kecil 2:.....	5
BAB 2.....	6
PENERAPAN ALGORITMA DIVIDE AND CONQUER.....	6
1. Algoritma Divide and Conquer.....	6
2. Implementasi Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree.....	6
BAB 3.....	9
SOURCE CODE.....	9
1. Struktur Proyek.....	9
2. ErrorCalculation.cpp.....	10
3. Image.cpp.....	12
4. InputHandler.cpp.....	15
5. main.cpp.....	18
6. QuadNode.cpp.....	20
7. QuadTree.cpp.....	22
BAB 4.....	24
EKSPERIMEN.....	24
BAB 5.....	31
ANALISIS.....	31
1. Analisis Kompleksitas Waktu.....	31
2. Analisis Kompleksitas Ruang.....	31
3. Analisis Hasil Percobaan.....	31
LAMPIRAN.....	32

BAB 1

DESKRIPSI MASALAH

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (*Divide and Conquer*):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- a) **Metode perhitungan variansi:** pilih metode perhitungan variansi berdasarkan opsi yang tersedia pada *Tabel 1*.
- b) **Threshold variansi:** nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
- c) **Minimum block size:** ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai *Tabel 1*.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- ❖ Jika variansi **di atas threshold** (*cek kasus khusus untuk metode bonus*), ukuran blok lebih besar dari **minimum block size**, dan ukuran blok setelah dibagi menjadi empat **tidak kurang dari minimum block size**, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- ❖ Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

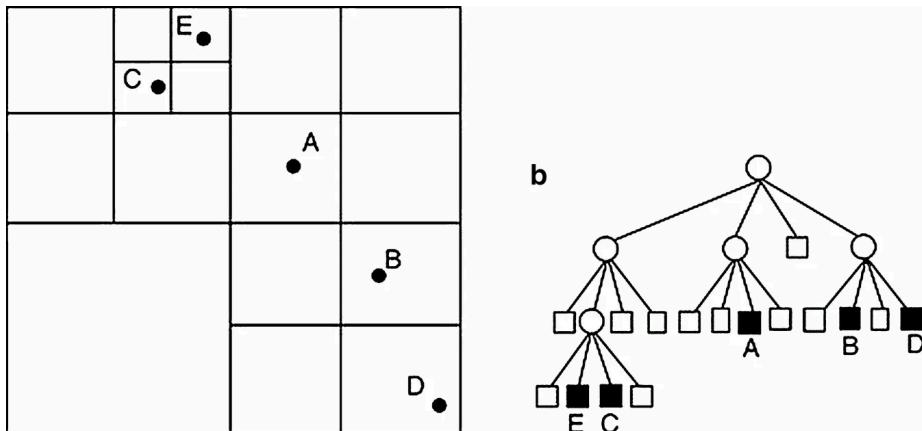
5. Rekursi dan Penghentian

Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- ❖ Error blok berada di bawah threshold.
- ❖ Ukuran blok setelah dibagi menjadi empat kurang dari *minimum block size*.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.



Gambar 3. Struktur Data Quadtree dalam Kompresi Gambar

(Sumber:

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

Parameter:

1. Error Measurement Methods (Metode Pengukuran Error)

Metode yang digunakan untuk menentukan seberapa besar perbedaan dalam satu blok gambar. Jika error dalam blok melebihi ambas batas (*threshold*), maka blok akan dibagi menjadi empat bagian yang lebih kecil.

Tabel 1. Metode Pengukuran Error

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

	N = Banyaknya piksel dalam satu blok
Mean Absolute Deviation (MAD)	$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $
	$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$
	MAD_c = Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk kanal warna c
Max Pixel Difference	μ_c = Nilai rata-rata tiap piksel dalam satu blok
	N = Banyaknya piksel dalam satu blok
	$D_c = \max(P_{i,c}) - \min(P_{i,c})$
	$D_{RGB} = \frac{D_R + D_G + D_B}{3}$
Entropy	D_c = Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
	$P_{i,c}$ = Nilai piksel pada posisi i untuk channel warna c
	$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$
	$H_{RGB} = \frac{H_R + H_G + H_B}{3}$
[Bonus]	H_c = Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
	$P_c(i)$ = Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)
	$SSIM_c(x, y) = \frac{(2\mu_{x,c}\mu_{y,c} + C_1)(2\sigma_{xy,c} + C_2)}{(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1)(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2)}$
	$SSIM_{RGB} = w_R \cdot SSIM_R + w_G \cdot SSIM_G + w_B \cdot SSIM_B$
<u>Structural Similarity Index (SSIM)</u> <u>(Referensi tambahan)</u>	Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Silakan lakukan eksplorasi untuk memahami serta memperoleh nilai konstanta pada formula SSIM, asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

2. Threshold (Ambang Batas)

Threshold adalah nilai batas yang menentukan apakah sebuah blok dianggap cukup seragam untuk disimpan atau harus dibagi lebih lanjut.

3. Minimum Block Size (Ukuran Blok Minimum)

Minimum block size (luas piksel) adalah ukuran terkecil dari sebuah blok yang diizinkan dalam proses kompresi. Jika ukuran blok yang akan dibagi menjadi empat sub-blok berada di bawah ukuran minimum yang telah dikonfigurasi, maka blok tersebut tidak akan dibagi lebih lanjut, meskipun error masih di atas threshold.

4. Compression Percentage (Persentase Kompresi) [BONUS]

Presentasi kompresi menunjukkan seberapa besar ukuran gambar berkurang dibandingkan dengan dengan ukuran aslinya setelah dikompresi menggunakan metode quadtree.

$$\text{Persentase Kompresi} = \left(1 - \frac{\text{Ukuran Gambar Terkompresi}}{\text{Ukuran Gambar Asli}}\right) \times 100\%$$

Spesifikasi Tugas Kecil 2:

- Buatlah program sederhana dalam bahasa **C/C#/C++/Java** (CLI) yang mengimplementasikan **algoritma divide and conquer** untuk melakukan kompresi gambar berbasis *quadtree* yang mengimplementasikan **seluruh parameter** yang telah disebutkan sebagai *user input*.
- Alur program:
 1. [INPUT] **alamat absolut** gambar yang akan dikompresi.
 2. [INPUT] metode perhitungan error (gunakan penomoran sebagai *input*).
 3. [INPUT] ambang batas (pastikan *range* nilai sesuai dengan metode yang dipilih).
 4. [INPUT] ukuran blok minimum.
 5. [INPUT] Target persentase kompresi (*floating number*, $1.0 = 100\%$), beri nilai 0 jika ingin menonaktifkan mode ini, jika mode ini aktif maka nilai threshold bisa menyesuaikan secara otomatis untuk memenuhi target persentase kompresi (bonus).
 6. [INPUT] **alamat absolut** gambar hasil kompresi.
 7. [INPUT] **alamat absolut** gif (bonus).
 8. [OUTPUT] waktu eksekusi.
 9. [OUTPUT] ukuran gambar sebelum.
 10. [OUTPUT] ukuran gambar setelah.
 11. [OUTPUT] persentase kompresi.
 12. [OUTPUT] kedalaman pohon.
 13. [OUTPUT] banyak simpul pada pohon.
 14. [OUTPUT] gambar hasil kompresi pada alamat yang sudah ditentukan.
 15. [OUTPUT] GIF proses kompresi pada alamat yang sudah ditentukan (bonus).

BAB 2

PENERAPAN ALGORITMA DIVIDE AND CONQUER

1. Algoritma Divide and Conquer

Algoritma Divide and Conquer merupakan pendekatan penyelesaian masalah dengan cara memecah masalah yang besar menjadi beberapa submasalah yang lebih kecil dan sederhana. Setelah disederhanakan, masing-masing dari tiap submasalah akan diselesaikan secara satu per satu dan secara rekursif. Setelah submasalah diselesaikan satu per satu, hasilnya akan digabung kembali untuk membentuk solusi dari permasalahan awal. Proses dari algoritma ini terdiri dari tiga tahap, yaitu *divide* (pembagian masalah), *conquer* (penyelesaian), dan *combine* (penyatuan kembali). Pendekatan ini biasanya digunakan untuk permasalahan yang memiliki pola berulang atau permasalahan yang dapat diselesaikan secara sistematis. Salah satu contoh penggunaan algoritma divide and conquer adalah pada algoritma *merge sort*, yaitu pengurutan bilangan.

Dalam konteks tugas kecil 2 ini, pendekatan algoritma divide and conquer akan digunakan untuk melakukan kompresi gambar dengan menggunakan metode quadtree. Gambar akan dibagi menjadi blok-blok yang lebih kecil (empat blok baru), lalu masing-masing blok akan dievaluasi berdasarkan nilai error dan nilai minimum dari block yang masih dapat diproses. Proses ini dilakukan secara rekursif hingga blok-blok dianggap cukup seragam ($\text{nilai error} < \text{threshold}$) atau ukurannya sudah terlalu kecil ($\text{blok} < \text{minBlockSize}$). Setelah dilakukan proses tersebut, gambar akan disusun ulang menjadi gambar baru yang sudah terkompresi.

2. Implementasi Algoritma Divide and Conquer dalam Kompresi Gambar dengan Metode Quadtree

```
Function build(image, threshold, minBlockSize, method):
    error = computeError(image, method)

    checkError = (error < threshold)
    checkSize = (width <= minBlockSize or height <= minBlockSize)

    If (checkError or checkSize):
        set isLeaf = true
        computeAverageColor(image)
        return

    set isLeaf = false

    halfWidth = width / 2
    halfHeight = height / 2

    children[0] = new QuadNode(x, y, halfWidth, halfHeight)
    children[1] = new QuadNode(x + halfWidth, y, width - halfWidth, halfHeight)
```

```

    children[2] = new QuadNode(x, y + halfHeight, halfWidth, height - halfHeight)
    children[3] = new QuadNode(x + halfWidth, y + halfHeight, width - halfWidth, height - halfHeight)

    for i = 0 to 3:
        children[i].build(image, threshold, minBlockSize, method)

End Function

```

Penjelasan Algoritma:

Fungsi build(image, threshold, minBlockSize, method) yang terdapat pada file QuadNode.cpp merupakan bagian dari program yang melakukan implementasi dengan algoritma divide and conquer untuk melakukan kompresi gambar. Prosesnya terdiri dari langkah-langkah berikut, yaitu

1. Divide (Pembagian)

Di awal, fungsi build akan menghitung error pada blok gambar dengan memanggil computerError. Nilai error ini akan menghitung seberapa banyak variasi dalam blok gambar. Setelah itu, akan dicek dua kondisi untuk memutuskan apakah blok gambar bisa dibagi lagi atau tidak dengan kondisi sebagai berikut:

- Apakah error lebih besar daripada threshold (ini berarti blok masih cukup beragam)
- Apakah ukuran blok (baik lebar, maupun tinggi) lebih besar dari minBlockSize (ini berarti blok masih cukup besar untuk bisa dibagi lagi)

Jika salah satu kondisi di atas tidak terpenuhi (ini berarti $\text{error} < \text{threshold}$ dan $\text{blok} < \text{minBlockSize}$), blok tersebut akan dianggap sudah seragam dan tidak perlu dibagi lagi, yakni akan menjadi leaf node. Akan tetapi, jika kedua kondisi di atas terpenuhi sama-sama, blok akan dibagi menjadi empat sub-blok.

2. Conquer (Penyelesaian)

Langkah berikutnya adalah setiap blok melakukan rekursi untuk setiap sub-blok yang baru dibuat. Untuk setiap anak node (sub-blok) yang baru dibuat, akan dipanggil lagi fungsi build. Untuk masing-masing sub-blok akan dilakukan langkah-langkah yang telah dijelaskan pada proses divide dan terus berulang hingga semua blok mencapai salah satu atau kedua dari kondisi-kondisi berikut:

- Error yang didapat di bawah threshold
- ukuran blok yang didapat di bawah ukuran minimum blok (minBlockSize)

3. Combine (Penggabungan)

Langkah terakhir adalah menggabungkan hasil dari langkah conquer. Setelah blok dibagi dan diproses lebih lanjut, hasil dari setiap sub-blok akan digabungkan menjadi gambar yang telah terkompresi. Setelah setiap sub-blok selesai diproses dan mencapai depth yang cukup (hingga menjadi leaf node), fungsi fill akan digunakan untuk mengisi warna rata-rata ke dalam blok gambar kompresi. Proses penggabungan ini akan terjadi secara rekursif untuk setiap node di Quadtree. Oleh karena itu, gambar

yang dihasilkan akan memiliki informasi warna yang disederhanakan pada area seragam dan area yang lebih kompleks tetap terjaga nilainya.

BAB 3

SOURCE CODE

1. Struktur Projek

Projek ini memiliki struktur direktori dan file sebagai berikut:

```
Tucil2_13523156/
└── .vscode/
    ├── bin/
    │   └── FreeImage.dll
    │   └── linux_compressor
    │   └── win_compressor.exe
    ├── doc/
    └── src/
        └── header
            ├── ErrorCalculation.hpp
            ├── FreeImage.h
            ├── Image.hpp
            ├── InputHandler.hpp
            ├── QuadNode.hpp
            └── QuadTree.hpp
        └── library
            └── FreeImage.lib
        ├── ErrorCalculation.cpp
        ├── Image.cpp
        ├── InputHandler.cpp
        ├── main.cpp
        ├── QuadNode.cpp
        └── QuadTree.cpp
    └── test/
        └── result/
    └── tc/
    └── Makefile
└── README.md
```

Keterangan

- .vscode/: Berisi file konfigurasi untuk editor Visual Studio Code.
- bin/: Berisi file yang dihasilkan setelah proses kompilasi.
 - a. FreeImage.dll: Berisi file dynamic-link library yang digunakan untuk memanfaatkan fungsi-fungsi yang disediakan oleh FreeImage.
 - b. linux_compressor: Berisi executable file untuk OS Linux.
 - c. win_compressor.exe: Berisi executable file untuk OS Windows
- doc/: Berisi laporan tugas kecil 2 dalam bentuk .pdf
- src/: Berisi kode sumber program
 - a. header: Berisi file header file (.hpp) yang digunakan untuk mendeklarasikan struktur data, fungsi, kelas, dan konstanta yang digunakan.
 - 1) ErrorCalculation.hpp: Berisi header file untuk menghitung error dengan parameter varians, MAD, max pixel difference.

- 2) FreeImage.h: Berisi header file untuk mengimpor fungsi dan tipe data dari library FreeImage.
- 3) Image.hpp: Berisi header file untuk memproses gambar.
- 4) InputHandler.hpp: Berisi header file untuk menangani input dari pengguna.
- 5) QuadNode.hpp: Berisi header file untuk mendefinisikan node dalam QuadTree.
- 6) QuadTree.hpp: Berisi header file untuk mendefinisikan QuadTree.
- b. library: Berisi file library dari FreeImage.lib
 - 1) FreeImage.lib: Berisi file library yang digunakan untuk menghubungkan fungsi-fungsi dari FreeImage.
- c. ErrorCalculation.cpp: Berisi file implementasi untuk menghitung error dengan parameter varians, MAD, max pixel difference.
- d. Image.cpp: Berisi file implementasi untuk memproses gambar.
- e. InputHandler.cpp: Berisi file implementasi untuk menangani input dari pengguna.
- f. main.cpp: Berisi file yang berfungsi sebagai entry point dari proyek
- g. QuadNode.cpp: Berisi file implementasi untuk mendefinisikan node dalam QuadTree.
- h. QuadTree.cpp: Berisi file implementasi untuk mendefinisikan QuadTree.
- test/: Berisi direktori testcase dan result.
 - a. testcase: Berisi file test untuk program.
 - b. result: Berisi file penyimpanan solusi dari testcase.
- Makefile: Berisi instruksi untuk build, clean, all, dan install.
- README.md : Berisi file readme yang mencakup deskripsi singkat, persyaratan, instalasi, kompilasi, cara menjalankan program dan nama author.

2. ErrorCalculation.cpp

```
#include "header/ErrorCalculation.hpp"
#include <cmath>
#include <vector>
#include <array>
#include <unordered_map>
using namespace std;

float ErrorCalculation::variance(const Image& image, int x, int y, int width, int height) {
    long long sumRed = 0, sumGreen = 0, sumBlue = 0;
    int total = width * height;
    for (int row = y; row < y + height; ++row) {
        for (int col = x; col < x + width; col++) {
            Pixel p = image.getPixel(col, row);
            sumRed += p.r;
        }
    }
}
```

```

        sumGreen += p.g;
        sumBlue += p.b;
    }
}
float meanRed = float(sumRed) / total;
float meanGreen = float(sumGreen) / total;
float meanBlue = float(sumBlue) / total;

float varRed = 0, varGreen = 0, varBlue = 0;
for (int row = y; row < y + height; row++) {
    for (int col = x; col < x + width; col++) {
        Pixel p = image.getPixel(col, row);
        varRed += (p.r - meanRed) * (p.r - meanRed);
        varGreen += (p.g - meanGreen) * (p.g - meanGreen);
        varBlue += (p.b - meanBlue) * (p.b - meanBlue);
    }
}
return (varRed + varGreen + varBlue) / (3 * total);
}

float ErrorCalculation::mad(const Image& image, int x, int y, int width, int height) {
    long long sumRed = 0, sumGreen = 0, sumBlue = 0;
    int total = width * height;
    for (int row = y; row < y + height; row++)
        for (int col = x; col < x + width; col++) {
            Pixel p = image.getPixel(col, row);
            sumRed += p.r;
            sumGreen += p.g;
            sumBlue += p.b;
        }
    float meanRed = float(sumRed) / total;
    float meanGreen = float(sumGreen) / total;
    float meanBlue = float(sumBlue) / total;

    float devRed = 0, devGreen = 0, devBlue = 0;
    for (int row = y; row < y + height; row++)
        for (int col = x; col < x + width; col++) {
            Pixel p = image.getPixel(col, row);
            devRed += abs(p.r - meanRed);
            devGreen += abs(p.g - meanGreen);
            devBlue += abs(p.b - meanBlue);
        }
    return (devRed + devGreen + devBlue) / (3 * total);
}

float ErrorCalculation::maxDiff(const Image& image, int x, int y, int width, int height) {
    int minRed = 255, minGreen = 255, minBlue = 255;
    int maxRed = 0, maxGreen = 0, maxBlue = 0;
    for (int row = y; row < y + height; row++)
        for (int col = x; col < x + width; col++) {
            Pixel p = image.getPixel(col, row);
            minRed = min(minRed, int(p.r));
            minGreen = min(minGreen, int(p.g));
            minBlue = min(minBlue, int(p.b));
        }
}

```

```

        maxRed = max(maxRed, int(p.r));
        maxGreen = max(maxGreen, int(p.g));
        maxBlue = max(maxBlue, int(p.b));
    }
    return (maxRed - minRed + maxGreen - minGreen + maxBlue - minBlue) / 3.0f;
}

float ErrorCalculation::entropy(const Image& image, int x, int y, int width, int height) {
    std::array<int, 256> freqRed = {}, freqGreen = {}, freqBlue = {};
    int total = width * height;

    for (int row = y; row < y + height; row++) {
        for (int col = x; col < x + width; col++) {
            Pixel p = image.getPixel(col, row);
            freqRed[p.r]++;
            freqGreen[p.g]++;
            freqBlue[p.b]++;
        }
    }

    auto entropy_channel = [&](const std::array<int, 256>& freq) {
        float H = 0.0f;
        for (int f : freq) {
            if (f > 0) {
                float p = float(f) / total;
                H -= p * log2(p);
            }
        }
        return H;
    };

    return (entropy_channel(freqRed) + entropy_channel(freqGreen) +
    entropy_channel(freqBlue)) / 3.0f;
}

```

File ErrorCalcultion.cpp berisi implementasi ErrorCalculation.hpp dari empat metode perhitungan error yang akan digunakan mengevaluasi nilai error, sehingga bisa diputuskan apakah blok akan dibagi lagi atau tidak. Terdapat empat metode yang diimplementasikan, yaitu **Variance**, **Mean Absolute Deviation (MAD)**, **Maximum Pixel Difference (MaxDiff)**, dan **Entropy**. Masing-masing dari metode akan menerima *input* berupa gambar, koordinat awal blok, dan ukuran blok yang akan dievaluasi.

3. Image.cpp

```

#include "header/Image.hpp"
#include "header/FreeImage.h"
#include <iostream>
#include <stdexcept>
using namespace std;

Image::Image() {
    width = 0;
}

```

```

height = 0;
data = nullptr;
}

Image::Image(int w, int h) {
    this->width = w;
    this->height = h;
    allocate(w, h);
}

Image::~Image() {
    deallocate();
}

bool Image::load(const string& path) {
    FreeImage_Initialise();

    FREE_IMAGE_FORMAT format = FreeImage_GetFileType(path.c_str(), 0);
    if (format == FIF_UNKNOWN) {
        format = FreeImage_GetFIFFFromFilename(path.c_str());
    }

    if (format == FIF_UNKNOWN) {
        cerr << "Format Gambar Tidak Diketahui " << path << endl; // debugging
        return false;
    }

    FIBITMAP* bitmap = FreeImage_Load(format, path.c_str());

    if (!bitmap) {
        cerr << "Tidak Dapat Memuat Gambar: " << path << endl; // debugging
        return false;
    }

    FIBITMAP* rgb = FreeImage_ConvertTo24Bits(bitmap);
    FreeImage_Unload(bitmap);

    this->width = FreeImage_GetWidth(rgb);
    this->height = FreeImage_GetHeight(rgb);
    BYTE* bits = FreeImage_GetBits(rgb);

    allocate(this->width, this->height);

    for (int row = 0; row < this->height; row++) {
        for (int col = 0; col < this->width; col++) {
            BYTE* pixel = bits + (row * FreeImage_GetPitch(rgb)) + (col * 3);
            data[row][col] = { pixel[2], pixel[1], pixel[0] }; // RGB order
        }
    }

    FreeImage_Unload(rgb);
    FreeImage_DeInitialise();
    return true;
}

```

```

bool Image::save(const string& path) const {
    FreeImage_Initialise();

    FIBITMAP* bitmap = FreeImage_Allocate(this->width, this->height, 24);
    if (!bitmap) {
        cerr << "Tidak Dapat Mengalokasikan Gambar: " << path << endl; // debugging
        return false;
    }

    for (int row = 0; row < this->height; height; row++) {
        for (int col = 0; col < this->width; width; col++) {
            RGBQUAD color;
            color.rgbRed = data[row][col].r;
            color.rgbGreen = data[row][col].g;
            color.rgbBlue = data[row][col].b;
            FreeImage_SetPixelColor(bitmap, col, row, &color);
        }
    }

    FREE_IMAGE_FORMAT format = FreeImage_GetFIFFFromFilename(path.c_str());
    if (!FreeImage_Save(format, bitmap, path.c_str(), 0)) {
        cerr << "Tidak Dapat Menyimpan Gambar: " << path << endl; // debugging
        FreeImage_Unload(bitmap);
        FreeImage_DeInitialise();
        return false;
    }

    FreeImage_Unload(bitmap);
    FreeImage_DeInitialise();
    return true;
}

void Image::allocate(int w, int h) {
    this->width = w;
    this->height = h;
    data = new Pixel*[this->height];
    for (int i = 0; i < this->height; height; i++) {
        data[i] = new Pixel[this->width];
    }
}

void Image::deallocate() {
    if (data) {
        for (int i = 0; i < this->height; height; i++) {
            delete[] data[i];
        }
        delete[] data;
        data = nullptr;
    }
}

Pixel Image::getPixel(int x, int y) const {
    return data[y][x];
}

```

```

}

void Image::setPixel(int x, int y, const Pixel& p) {
    data[y][x] = p;
}

int Image::getWidth() const {
    return this->width;
}

int Image::getHeight() const {
    return this->height;
}

void Image::free() {
    deallocate();
}

```

File Image.cpp berisi implementasi dari Image.hpp yang dapat memuat, memanipulasi, dan menyimpan gambar dalam format piksel RGB. File ini menggunakan library eksternal, yaitu FreeImage. Fungsi load() digunakan untuk memuat gambar dari file yang kemudian akan dikonversikan ke format 24 bit RGB dan disimpan dalam bentuk matriks dua dimensi. Matriks tersebut menyimpan nilai merah, hijau, dan biru dari setiap piksel. Fungsi save() digunakan untuk menyimpan gambar ke file (menggunakan *absolute path*) sesuai dengan tujuan. Selama prosesnya, nilai piksel dari matriks akan dituliskan kembali ke objek FIBITMAP dari FreeImage, baru disimpan ke file. allocate() dan deallocate() berfungsi untuk mengatur alokasi dan dealokasi matriks piksel. Terdapat juga fungsi getWidth() dan getHeight() untuk mendapatkan dimensi gambar.

4. InputHandler.cpp

```

#include "header/InputHandler.hpp"
#include <iostream>
#include <sstream>
#include <filesystem>
namespace fs = std::filesystem;
using namespace std;

InputHandler::InputHandler() {

}

bool InputHandler::checkValid() {
    bool valid = true;
    if (!fs::exists(inputPath)) {
        cerr << "Path input tidak valid" << endl;
        return false;
    }
    if (outputPath.empty() || !fs::path(outputPath).has_extension()) {
        cout << "Path output tidak valid" << endl;
    }
}

```

```

        valid = false;
    }
    if (method < 1 || method > 4) {
        cerr << "Metode tidak valid" << endl;
        valid = false;
    }
    if (threshold < 0) {
        cerr << "Threshold tidak valid. Masukkan angka >= 0." << endl;
        valid = false;
    }
    if (minBlockSize <= 0) {
        cerr << "Ukuran minimum blok tidak valid. Masukkan angka > 0." << endl;
        valid = false;
    }
    return valid;
}

void InputHandler::InputSingleLine() {
    string line;
    while (true) {
        cout << "Pilihan 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 input dalam satu baris (<input path> <output path> <Metode>
<threshold> <minBlockSize>): ";
        getline(cin, line);
        stringstream ss(line);
        ss >> inputPath >> outputPath >> method >> threshold >> minBlockSize;
        if (checkValid()) {
            break;
        }
        cout << endl << "=====Silakan coba lagi=====" << endl;
    }
}

void InputHandler::InputOnebyOne() {
    while (true) {
        while (true) {
            cout << "Masukkan path gambar input: ";
            getline(cin, inputPath);
            if (fs::exists(inputPath)) break;
            cout << "File input tidak valid" << endl;
        }

        while (true) {
            cout << "Masukkan path gambar output: ";
            getline(cin, outputPath);
            if (!outputPath.empty() && fs::path(outputPath).has_extension()) break;
            cout << "Path output tidak valid" << endl;
        }

        string input;

```

```

while (true) {
    cout << "Pilihan 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 << "Pilih metode error: ";
    getline(cin, input);
    stringstream ss(input);
    if (ss >> method && method >= 1 && method <= 4) break;
    cout << "Metode tidak valid. Masukkan angka antara 1 sampai 4.\n";
}

while (true) {
    cout << "Masukkan threshold (>= 0): ";
    getline(cin, input);
    stringstream ss(input);
    if (ss >> threshold && threshold >= 0) break;
    cout << "Threshold tidak valid. Masukkan angka >= 0.\n";
}

while (true) {
    cout << "Masukkan minimum block size (> 0): ";
    getline(cin, input);
    stringstream ss(input);
    if (ss >> minBlockSize && minBlockSize > 0) break;
    cout << "Ukuran minimum blok tidak valid. Masukkan angka > 0.\n";
}
break;
}

string InputHandler::getInputPath() const {
    return inputPath;
}

string InputHandler::getOutputPath() const {
    return outputPath;
}

int InputHandler::getMethod() const {
    return method;
}

float InputHandler::getThreshold() const {
    return threshold;
}

int InputHandler::getMinBlockSize() const {
    return minBlockSize;
}

```

File InputHandler.cpp berisi implementasi dari InputHandler.hpp untuk menangani proses *input* dari pengguna. Terdapat dua cara untuk menerima *input*, yaitu menggunakan InputSingleLine(), yang berarti *input* dalam satu baris sekaligus, atau dengan menggunakan InputOnebyOne(), yang berarti *input* secara satu per satu. Parameter *input* berupa *absolute path* dari file gambar *input*, *absolute path* dari file gambar *output*, metode pengukuran, nilai threshold, dan ukuran minimum blok yang dapat dibagi. Setiap *input* juga akan dilakukan validasi berdasarkan masukan yang diberikan.

5. main.cpp

```
#include <iostream>
#include <chrono>
#include <sstream>
#include <filesystem>
#include "header/InputHandler.hpp"
#include "header/Image.hpp"
#include "header/QuadTree.hpp"

namespace fs = std::filesystem;
using namespace std;

int main(int argc, char** argv) {
    InputHandler io;

    try {
        int choice;
        while (true) {
            cout << "===== INPUT =====" << endl;
            cout << "Pilih Cara untuk Input: " << endl;
            cout << "1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)" << endl;
            cout << "2. Input Satu per Satu" << endl;
            cout << "Pilihan: ";

            string line;
            getline(cin, line);
            stringstream ss(line);
            if (!(ss >> choice)) {
                cout << "Input harus berupa angka!\n";
                continue;
            }

            if (choice == 1) {
                io.InputSingleLine();
                break;
            } else if (choice == 2) {
                io.InputOnebyOne();
                break;
            } else {
                cerr << "Pilihan tidak valid." << endl;
            }
        }
    }
}
```

```

Image img;
if (!img.load(io.getInputPath())) {
    cerr << "Gagal Memuat Gambar" << endl;
    return 1;
}

uintmax_t sizeBefore = fs::file_size(io.getInputPath());

cout << "Gambar dimuat: " << img.getWidth() << "x" << img.getHeight() << endl;

auto start = chrono::high_resolution_clock::now();

cout << "Memulai kompresi..." << endl; // debugging
QuadTree QT(img, io.getThreshold(), io.getMinBlockSize(), io.getMethod());
cout << "Kompresi selesai." << endl; // debugging

Image compressed(img.getWidth(), img.getHeight());
QT.fill(compressed);

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

if (!compressed.save(io.getOutputPath())) {
    cerr << "Gagal Menyimpan Gambar" << endl;
    return 1;
}

uintmax_t sizeAfter = fs::file_size(io.getOutputPath());

double compressionPercentage = (1.0 - static_cast<double>(sizeAfter) / sizeBefore)
* 100.0;

cout << endl << "===== OUTPUT =====" << endl;
cout << "Gambar disimpan: " << io.getOutputPath() << endl;
cout << "Ukuran gambar sebelum kompresi: " << sizeBefore << " bytes" << endl;
cout << "Ukuran gambar setelah kompresi: " << sizeAfter << " bytes" << endl;
cout << "Persentase kompresi: " << compressionPercentage << "%" << endl;
cout << "Waktu kompresi: " << duration << " ms" << endl;
cout << "Kedalaman QuadTree: " << QT.getDepth() << endl;
cout << "Jumlah node: " << QT.countNodes() << endl;
} catch (const exception& e) {
    cerr << "Terjadi kesalahan: " << e.what() << endl;
    return 1;
}
return 0;
}

```

File main.cpp berfungsi sebagai *entry point* dari program. Program dimulai dengan menampilkan pilihan metode *input* yang bisa dipilih oleh pengguna (apakah sekaligus dalam satu baris atau satu per satu). Parameter *input* berupa *absolute path* dari file gambar *input*, *absolute path* dari file gambar *output*, metode pengukuran, nilai threshold, dan ukuran

minimum blok yang dapat dibagi yang ditangani oleh InputHandler. Setelah menerima *input*, akan dilakukan pemuatan gambar dengan menggunakan Image. Jika berhasil, proses kompresi akan dimulai. Proses kompresi akan dilakukan dengan membuat objek QuadTree yang akan menerima gambar, threshold, ukuran minimum block dan metode error. Setelah itu, akan dilakukan proses secara rekursif sesuai dengan penjelasan yang telah diterangkan sebelumnya di bagian penerapan algoritma divide and conquer. Gambar yang telah dikompresi akan disimpan. Lalu, akan ditampilkan *output* sesuai dengan permintaan dari spesifikasi tugas kecil 2 ini.

6. QuadNode.cpp

```
#include "header/QuadNode.hpp"
#include "header/ErrorCalculation.hpp"
#include <algorithm>

QuadNode::QuadNode(int x, int y, int width, int height)
    : x(x), y(y), width(width), height(height), isLeaf(true) {
    for (int i = 0; i < 4; ++i) {
        children[i] = nullptr;
    }
}

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

void QuadNode::computeAverageColor(const Image& image) {
    long sumR = 0, sumG = 0, sumB = 0;
    for (int row = y; row < y + height; ++row) {
        for (int col = x; col < x + width; ++col) {
            Pixel p = image.getPixel(col, row);
            sumR += p.r;
            sumG += p.g;
            sumB += p.b;
        }
    }
    int total = width * height;
    color.r = sumR / total;
    color.g = sumG / total;
    color.b = sumB / total;
}

float QuadNode::computeError(const Image& image, int method) {
    switch (method) {
        case 1:
            return ErrorCalculation::variance(image, x, y, width, height);
        case 2:
            return ErrorCalculation::mad(image, x, y, width, height);
        case 3:
            return ErrorCalculation::maxDiff(image, x, y, width, height);
    }
}
```

```

        case 4:
            return ErrorCalculation::entropy(image, x, y, width, height);
        default:
            return 0.0f;
    }
}

void QuadNode::build(const Image& image, float threshold, int minBlockSize, int
method) {
    float error = computeError(image, method);

    bool checkError = error < threshold;
    bool checkSize = (width <= minBlockSize || height <= minBlockSize);

    if (checkError || checkSize) {
        isLeaf = true;
        computeAverageColor(image);
        return;
    }

    isLeaf = false;
    int halfWidth = width / 2;
    int halfHeight = height / 2;

    children[0] = new QuadNode(x, y, halfWidth, halfHeight);
    children[1] = new QuadNode(x + halfWidth, y, width - halfWidth, halfHeight);
    children[2] = new QuadNode(x, y + halfHeight, halfWidth, height - halfHeight);
    children[3] = new QuadNode(x + halfWidth, y + halfHeight, width - halfWidth, height
- halfHeight);

    for (int i = 0; i < 4; ++i) {
        children[i]->build(image, threshold, minBlockSize, method);
    }
}

void QuadNode::fill(Image& output) const {
    if (isLeaf) {
        for (int row = y; row < y + height; ++row) {
            for (int col = x; col < x + width; ++col) {
                output.setPixel(col, row, color);
            }
        }
    } else {
        for (int i = 0; i < 4; ++i) {
            if (children[i]) {
                children[i]->fill(output);
            }
        }
    }
}

int QuadNode::getDepth() const {
    if (isLeaf) return 0;
    int maxDepth = 0;

```

```

        for (int i = 0; i < 4; ++i) {
            if (children[i]) {
                maxDepth = std::max(maxDepth, children[i]->getDepth());
            }
        }
        return maxDepth + 1;
    }

int QuadNode::countNodes() const {
    if (isLeaf) return 1;
    int count = 1;
    for (int i = 0; i < 4; ++i) {
        if (children[i]) {
            count += children[i]->countNodes();
        }
    }
    return count;
}

```

File QuadNode.cpp berisi struktur data dari QuadTree yang digunakan untuk membagi gambar menjadi bagian-bagian (sub-blok) yang lebih kecil. Pada file ini terdapat computeAverageColor, yaitu fungsi untuk menghitung warna rata-rata dari piksel-piksel dalam blok gambar yang ditentukan oleh koordinat x, y dan dimensi lebar dan tinggi dari blok. Fungsi computeError digunakan untuk menghitung error pada suatu blok gambar berdasarkan metode yang dipilih. Build merupakan fungsi inti yang menjalankan algoritma divide and conquer seperti yang telah dijelaskan pada BAB 2. fill merupakan fungsi yang digunakan untuk mengisi setiap blok di dalam Quadtree yang merupakan lead node dengan warna rata-rata dari blok tersebut. Selain itu, terdapat juga fungsi getDepth dan countNodes yang akan menghasilkan kedalaman dan node dari proses kompresi ini.

7. QuadTree.cpp

```

#include "header/Quadtree.hpp"

QuadTree::QuadTree(const Image& image, float threshold, int minBlockSize, int method)
{
    width = image.getWidth();
    height = image.getHeight();
    root = new QuadNode(0, 0, width, height);
    root->build(image, threshold, minBlockSize, method);
}

QuadTree::~QuadTree() {
    delete root;
}

void QuadTree::fill(Image& output) const {
    root->fill(output);
}

```

```
int QuadTree::getDepth() const {
    return root->getDepth();
}

int QuadTree::countNodes() const {
    return root->countNodes();
}
```

Berisi kelas dan fungsi dari QuadTree. Disini lah entry point pertama ketika program akan mulai melakukan kompresi gambar dengan metode QuadTree.

BAB 4

EKSPERIMENT

Uji coba akan dilakukan dalam berbagai bentuk metode perhitungan error dan menggunakan file dengan ekstensi .jpg, .png, dan .jpeg.

● Test Case 1

Input:

```
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc1.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc1.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 1
Masukkan threshold (>= 0): 2
Masukkan minimum block size (> 0): 2
Gambar dimuat: 640x427
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc1.jpg
Ukuran gambar sebelum kompresi: 23754 bytes
Ukuran gambar setelah kompresi: 16642 bytes
Persentase kompresi: 29.9402%
Waktu kompresi: 56 ms
Kedalaman QuadTree: 8
Jumlah node: 44421
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156>
```

Output yang Dihasilkan:



- **Test Case 2**

Input:

```
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc2.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc2.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 2
Masukkan threshold (>= 0): 4
Masukkan minimum block size (> 0): 4
Gambar dimuat: 640x426
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc2.jpg
Ukuran gambar sebelum kompresi: 81083 bytes
Ukuran gambar setelah kompresi: 38673 bytes
Persentase kompresi: 52.3044%
Waktu kompresi: 50 ms
Kedalaman QuadTree: 7
Jumlah node: 14385
```

Output yang Dihasilkan:



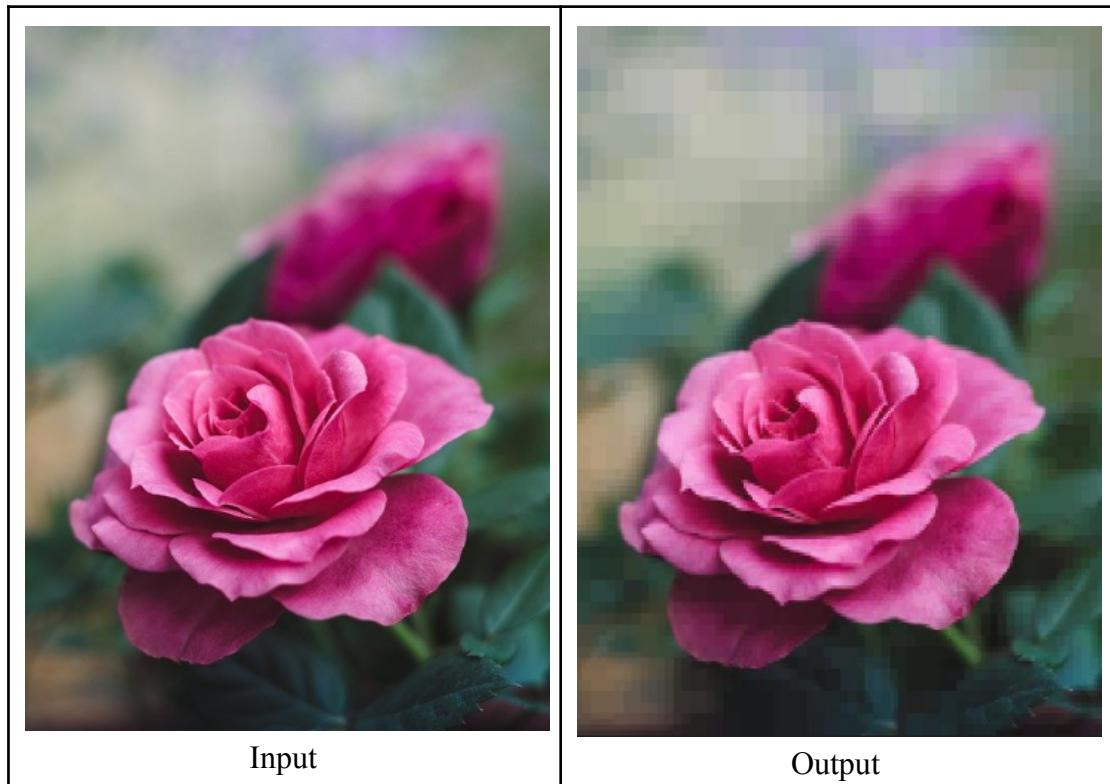
- **Test Case 3**

Input:

```
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc3.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc3.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 3
Masukkan threshold (>= 0): 20
Masukkan minimum block size (> 0): 2
Gambar dimuat: 480x640
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc3.jpg
Ukuran gambar sebelum kompresi: 40740 bytes
Ukuran gambar setelah kompresi: 30723 bytes
Persentase kompresi: 24.5876%
Waktu kompresi: 68 ms
Kedalaman QuadTree: 8
Jumlah node: 25329
```

Output yang Dihasilkan:



- **Test Case 4**

Input:

```
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc4.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc4.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 4
Masukkan threshold (>= 0): 4
Masukkan minimum block size (> 0): 4
Gambar dimuat: 640x426
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc4.jpg
Ukuran gambar sebelum kompresi: 70055 bytes
Ukuran gambar setelah kompresi: 43062 bytes
Persentase kompresi: 38.5312%
Waktu kompresi: 105 ms
Kedalaman QuadTree: 7
Jumlah node: 14605
```

Output yang Dihasilkan:



- **Test Case 5**

Input:

```

Masukkan threshold (>= 0): 4
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc5.png
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc5.png
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 1
Masukkan threshold (>= 0): 40
Masukkan minimum block size (> 0): 4
Gambar dimuat: 292x173
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc5.png
Ukuran gambar sebelum kompresi: 11047 bytes
Ukuran gambar setelah kompresi: 8092 bytes
Persentase kompresi: 26.7493%
Waktu kompresi: 7 ms
Kedalaman QuadTree: 6
Jumlah node: 2577

```

Output yang Dihasilkan:



- **Test Case 6**

Input:

```

Masukkan threshold (>= 0): 40
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc6.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc6.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 2
Masukkan threshold (>= 0): 8
Masukkan minimum block size (> 0): 4
Gambar dimuat: 275x183
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc6.jpg
Ukuran gambar sebelum kompresi: 9168 bytes
Ukuran gambar setelah kompresi: 9946 bytes
Persentase kompresi: -8.48604%
Waktu kompresi: 14 ms
Kedalaman QuadTree: 6
Jumlah node: 4741

```

Output yang Dihasilkan:



- **Test Case 7**

Input:

```
Masukkan threshold (>= 0): 100
PS D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156> .\bin\win_compressor
===== INPUT =====
Pilih Cara untuk Input:
1. Input dalam Satu Baris Sekaligus dengan format (<input path> <output path> <Metode> <threshold> <minBlockSize>)
2. Input Satu per Satu
Pilihan: 2
Masukkan path gambar input: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\tc\tc7.jpg
Masukkan path gambar output: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc7.jpg
Pilihan Metode Pengukuran Error:
1. Variance
2. Mean Absolute Deviation (MAD)
3. Max Pixel Difference
4. Entropy
Pilih metode error: 3
Masukkan threshold (>= 0): 150
Masukkan minimum block size (> 0): 4
Gambar dimuat: 640x384
Memulai kompresi...
Kompresi selesai.

===== OUTPUT =====
Gambar disimpan: D:\Akademik\Tugas_Sem_4\Stima\Tucil\Tucil2_13523156\test\result\tc7.jpg
Ukuran gambar sebelum kompresi: 69837 bytes
Ukuran gambar setelah kompresi: 14270 bytes
Persentase kompresi: 79.5667%
Waktu kompresi: 30 ms
Kedalaman QuadTree: 7
Jumlah node: 1293
```

Output yang Dihasilkan:



BAB 5

ANALISIS

1. Analisis Kompleksitas Waktu

Pada program kompresi gambar menggunakan algoritma Divide and Conquer dengan metode QuadTree, kompleksitas waktu dipengaruhi oleh pembagian gambar menjadi blok yang lebih kecil dan perhitungan error serta rata-rata warna untuk setiap blok. Setiap blok dalam QuadTree dihitung error menggunakan metode yang telah ditetapkan dengan waktu yang dibutuhkan $O(w * h)$, dengan w dan h adalah lebar dan tinggi. Secara keseluruhan, kompleksitas waktu untuk menghitung error dan rata-rata warna untuk seluruh gambar adalah $O(N^2)$ dengan N adalah jumlah piksel dalam gambar. Pembagian gambar dilakukan secara rekursif dengan QuadTree memiliki kedalaman maksimum, yaitu $\log(N)$ dengan N adalah jumlah piksel dalam gambar. Meskipun pembagian dilakukan secara rekursif, setiap nilai piksel hanya dihitung beberapa kali dalam perhitungan error dan warna rata-rata. Dengan demikian, kompleksitas waktu total adalah $O(N^2)$.

2. Analisis Kompleksitas Ruang

Kompleksitas ruang pada program kompresi gambar ini didominasi oleh penyimpanan gambar dan penyimpanan QuadTree. Untuk menyimpan gambar, digunakan struktur data Image yang menyimpan setiap piksel dalam format RGB. Memori yang dibutuhkan untuk menyimpan gambar adalah $O(N^2)$ dengan N adalah jumlah piksel dalam gambar. Selain itu, QuadTree juga memerlukan ruang untuk menyimpan informasi mengenai setiap blok gambar, termasuk posisi (x,y), ukuran (width, height), warna rata-rata blok, dan referencee untuk empat sub-blok. Dalam kasus terburuk, QuadTree akan memiliki $O(N^2 / s^2)$ node, dengan s adalah ukuran blok terkecil yang ditentukan oleh parameter minBlockSize. Terakhir, dibutuhkan memori untuk menyimpan gambar hasil kompresi sebesar $O(N^2)$. Secara keseluruhan, total kompleksitas memori yang digunakan adalah $O(N^2)$.

3. Analisis Hasil Percobaan

Secara keseluruhan, algoritma Divide and Conquer dengan metode Quadtree efektif dalam mengkompresi gambar dengan menjaga keseimbangan antara kualitas gambar dan ukuran file. Threshold dan minBlockSize merupakan parameter yang sangat penting dalam menentukan hasil kompresi gambar dan waktu eksekusi. Dengan nilai threshold yang rendah, blok dapat membagi sub-blok dengan lebih banyak dan menghasilkan kompresi yang cukup baik. Hal ini tentu berbeda jika threshold memiliki nilai yang cukup tinggi, hasil kompresi memiliki ukuran yang lebih kecil, tetapi kualitas gambar yang dihasilkan lebih rendah dari asilnya. Untuk minBlockSize, semakin besar minBlockSize, maka semakin sedikit pula divide yang akan dilakukan.

LAMPIRAN

- Pranala Repository:
https://github.com/Inforable/Tucil2_13523156

- Tabel Pengecekan Fitur

Poin	Ya	Tidak
1. Program berhasil dikompilasi tanpa kesalahan	<input checked="" type="checkbox"/>	
2. Program berhasil dijalankan	<input checked="" type="checkbox"/>	
3. Program berhasil melakukan kompresi gambar sesuai parameter yang ditentukan	<input checked="" type="checkbox"/>	
4. Mengimplementasi seluruh metode perhitungan error wajib	<input checked="" type="checkbox"/>	
5. [Bonus] Implementasi persentase kompresi sebagai parameter tambahan		<input checked="" type="checkbox"/>
6. [Bonus] Implementasi Structural Similarity Index (SSIM) sebagai metode pengukuran error		<input checked="" type="checkbox"/>
7. [Bonus] Output berupa GIF Visualisasi Proses pembentukan Quadtree dalam Kompresi Gambar		<input checked="" type="checkbox"/>
8. Program dan laporan dibuat (kelompok) sendiri	<input checked="" type="checkbox"/>	