Laporan Tugas Kecil 2

IF2211 Strategi Algoritma

Kompresi Gambar Dengan Metode Quadtree

Oleh:
Muhammad Aditya Rahmadeni
13523028

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

BAB I
Deksripsi Masalah

Quadtree adalah struktur data hierarkis yang digunakan untuk membagi ruang atau data menjadi bagian yang lebih kecil, yang sering digunakan dalam pengolahan gambar. Dalam konteks kompresi gambar, Quadtree membagi gambar menjadi blok-blok kecil berdasarkan keseragaman warna atau intensitas piksel. Prosesnya dimulai dengan membagi gambar menjadi empat bagian, lalu memeriksa apakah setiap bagian memiliki nilai yang seragam berdasarkan analisis sistem warna RGB, yaitu dengan membandingkan komposisi nilai merah (R), hijau (G), dan biru (B) pada piksel-piksel di dalamnya. Jika bagian tersebut tidak seragam, maka bagian tersebut akan terus dibagi hingga mencapai tingkat keseragaman tertentu atau ukuran minimum yang ditentukan.

Dalam implementasi teknis, sebuah Quadtree direpresentasikan sebagai simpul (node) dengan maksimal empat anak (children). Simpul daun (leaf) merepresentasikan area gambar yang seragam, sementara simpul internal menunjukkan area yang masih membutuhkan pembagian lebih lanjut. Setiap simpul menyimpan informasi seperti posisi (x, y), ukuran (width, height), dan nilai rata-rata warna atau intensitas piksel dalam area tersebut. Struktur ini memungkinkan pengkodean data gambar yang lebih efisien dengan menghilangkan redundansi pada area yang seragam. QuadTree sering digunakan dalam algoritma kompresi lossy karena mampu mengurangi ukuran file secara signifikan tanpa mengorbankan detail penting pada gambar.

BAB II
Algoritma Divide and Conquer

Pendekatan *divide and conquer* yang digunakan dalam kompresi gambar berbasis Quadtree bertujuan untuk membagi gambar menjadi blok-blok kecil secara rekursif, kemudian menyederhanakan atau menormalkan blok-blok tersebut apabila tingkat variasi warnanya (error) cukup rendah. Dengan demikian, algoritma ini dapat mengurangi kompleksitas data tanpa mengorbankan terlalu banyak informasi visual.

Langkah-langkah algoritma secara umum dijelaskan sebagai berikut:

1. **Perhitungan Error pada Blok Gambar**
   Langkah awal adalah menghitung nilai error pada suatu blok gambar. Error ini mencerminkan tingkat ketidakhomogenan warna pada blok tersebut. Semakin besar nilai error, maka semakin bervariasi warna dalam blok. Perhitungan ini dapat dilakukan menggunakan metode statistik, seperti variansi atau deviasi standar terhadap nilai warna piksel dalam blok.

2. **Pemeriksaan Ukuran Minimum dan Threshold Error**
   Setelah error dihitung, dilakukan pengecekan terhadap dua kondisi:

   - Apakah ukuran blok lebih kecil atau sama dengan ukuran minimum yang telah ditentukan.
   - Apakah nilai error lebih kecil atau sama dengan nilai ambang batas (*threshold*) yang ditentukan.

   Jika salah satu kondisi tersebut terpenuhi, maka blok dianggap cukup homogen atau tidak layak untuk dibagi lebih lanjut.

3. **Normalisasi Blok Gambar**
   Apabila proses pembagian dihentikan, maka blok gambar akan dinormalisasi. Proses normalisasi dilakukan dengan cara menyelaraskan nilai-nilai warna di dalam blok agar tersebar secara proporsional dalam rentang 0 hingga 255. Tujuan dari langkah ini adalah untuk mengurangi variasi warna internal sambil mempertahankan tampilan visual yang seragam.

4. **Pembagian Blok menjadi Empat Sub-Blok**
   Jika blok masih cukup besar dan nilai error melebihi ambang batas, maka blok tersebut akan dibagi menjadi empat bagian (sub-blok) secara merata, yaitu:

   - Kuadran kiri atas
   - Kuadran kanan atas

- Kuadran kiri bawah
- Kuadran kanan bawah

Titik tengah dari blok digunakan sebagai batas pembagian antara sub-blok satu dengan yang lain.

5. **Pemrosesan Rekursif untuk Setiap Sub-Blok**
   Setiap sub-blok kemudian diproses kembali menggunakan algoritma yang sama, dimulai dari perhitungan error hingga normalisasi atau pembagian lebih lanjut. Proses ini dilakukan secara rekursif hingga seluruh blok dalam gambar telah diproses.

Dengan pendekatan ini, bagian-bagian gambar yang homogen dapat disederhanakan secara efisien, sementara bagian-bagian dengan detail tinggi tetap dipertahankan dalam ukuran blok yang lebih kecil. Hal ini memungkinkan kompresi gambar dilakukan secara adaptif berdasarkan konten visualnya.

Pseudocode:

```
Function DivideAndConquer(image, startX, endX, startY, endY, threshold, minSize):

    error ← HitungError(image, startX, endX, startY, endY)

    If (endX - startX ≤ minSize OR endY - startY ≤ minSize) OR (error ≤ threshold) Then
        NormalisasiSubBlok(image, startX, endX, startY, endY)
        Return
    EndIf

    midX ← (startX + endX) / 2
    midY ← (startY + endY) / 2

    // Rekursif ke empat kuadran
    DivideAndConquer(image, startX, midX, startY, midY, threshold, minSize)     //
Kuadran kiri atas
    DivideAndConquer(image, midX+1, endX, startY, midY, threshold, minSize)     //
Kuadran kanan atas
    DivideAndConquer(image, startX, midX, midY+1, endY, threshold, minSize)     //
Kuadran kiri bawah
    DivideAndConquer(image, midX+1, endX, midY+1, endY, threshold, minSize)     //
Kuadran kanan bawah
EndFunction
```

BAB III
Source Program

Program utama terdiri dari 4 buah file sebagai hasil modularisasi dan satu file main sehingga program terdiri dari 5 file

- Main
- Quadtree
- ErrorCalculation
- Image

Setiap *code* ditampilkan dibawah ini

1. main.c

```
// main.c
// 13523028 - Muhammad Aditya Rahmadeni

#include <stdint.h>
#include <stdlib.h>
#include <string.h>
#include "header/quadtree.h"
#include <time.h>

char* absolPath;
Image image;
Image res;

void inputHandler()
{
    // [Input] Absolut Path dari file
    while(true)
    {
        printf("\033[1;33m[Input]\033[0m Masukkan nama file gambar yang ingin diproses (
\033[1;33mCATATAN : PASTIKAN FILE ADA DALAM FOLDER TEST ATAU PATH LENGKAP FILE ADA\033[0m ) \n> ");
        char* fileName = (char*)malloc(300 * sizeof(char));
        absolPath = (char*)malloc(500 * sizeof(char));

        strcpy(absolPath, "./test/");
        scanf("%300s", fileName);
        strcat(absolPath, fileName);

        FILE* file = fopen(absolPath, "r");
        if(file == NULL)
        {
            file = fopen(fileName, "r");
            if(file == NULL)
```

```
            {
                printf("\033[1;31m[ERROR]\033[0m File pada \033[1;34m%s\033[0m ataupun
\033[1;34m%s\033[0m tidak ditemukan, silahkan coba lagi.\n", fileName, absolPath);
                continue;
            }
            else
            {
                printf("\033[1;32m[Success]\033[0m File ditemukan !\n\n");
                extractImage(file, &image);
                break;
            }
        }
        else
        {
            printf("\n\033[1;32m[Success]\033[0m File ditemukan !\n\n");
            extractImage(file, &image);
            break;
        }
    }
    // [Input] Metode perhitungan error & Threshold
    chooseErrorMethod();
}



void processImage(Image* im) {
    printf("\033[1;32m[INFO]\033[0m Gambar akan mulai diproses...\n\n");

    // Timer Mulai
    clock_t start = clock();
    divideNConquer(im, threshold, 0, im->width, 0, im->height, minSize);
    clock_t end = clock();
    // Timer Selesai

    double timeTaken = (double)(end - start) / CLOCKS_PER_SEC;
    printf("\033[1;32m[INFO]\033[0m Gambar berhasil diproses dalam waktu \033[1;34m%.2f
detik\033[0m\n", timeTaken);
}

int main()
{
    inputHandler();

    // Copy image ke res untuk membandingkan nanti
    res.width = image.width;
    res.height = image.height;
    res.data = (uint8_t*)malloc(res.width * res.height * 3);
    memcpy(res.data, image.data, res.width * res.height * 3);
```

```
    // Proses gambar
    processImage(&res);

    char fullpath[300] = "./";
    saveImage(fullpath, &res);

    // Bersih bersih cihuy
    free(image.data);
    free(res.data);
}
```

2. Quadtree.h

```
// Header Quadtree
// 13523028 Muhammad Aditya Rahmadeni

#ifndef QUADTREE_H
#define QUADTREE_H

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include "image.h"
#include "errorcalculations.h"

/**
 * @brief Implementasi divide and conquer dari Quadtree
 * @param im Pointer ke struktur Image yang akan diproses
 * @param threshold Nilai threshold untuk pemisahan
 * @param startX Koordinat X awal
 * @param endX Koordinat X akhir
 * @param startY Koordinat Y awal
 * @param endY Koordinat Y akhir
 * @param minSize Ukuran minimum untuk pemisahan
 * @return void
 * @note Fungsi ini akan membagi gambar menjadi sub-gambar berdasarkan threshold dan ukuran minimum
 */
void divideNConquer(Image* im, double threshold, int startX, int endX, int startY, int endY, int
minSize);

/**
 * @brief Mengnormalisasi sub blok gambar
 *
 * @param im Pointer ke struktur Image yang akan diproses
```

```
 * @param startX Koordinat X awal
 * @param endX Koordinat X akhir
 * @param startY Koordinat Y awal
 * @param endY Koordinat Y akhir
 * @return void
 * @note Fungsi ini akan mengnormalisasi sub blok gambar berdasarkan nilai minimum dan maksimum
 */
void normalizeSubImage(Image* im, int startX, int endX, int startY, int endY);

#endif
```

3. Quadtree.c

```
#include "header/quadtree.h"

void normalizeSubImage(Image* im, int startX, int endX, int startY, int endY) {
    uint64_t sum[3] = {0, 0, 0};
    int total = 0;

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * im->width + x) * 3;
            for (int c = 0; c < 3; c++) {
                sum[c] += im->data[idx + c];
            }
            total++;
        }
    }

    uint8_t avg[3] = {
        sum[0] / total,
        sum[1] / total,
        sum[2] / total
    };

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * im->width + x) * 3;
            for (int c = 0; c < 3; c++) {
                im->data[idx + c] = avg[c];
            }
        }
    }
}
```

```
void divideNConquer(Image* im, double threshold, int startX, int endX, int startY, int endY, int
minSize) {
    // Hitung eror sesuai metoed yang sudah dipilih
    double error = calculateError(im->data, im->width, im->height, startX, endX, startY, endY);

    // Cek apakah ukuran sesuai batas yang ditentukan atau eror masih diatas batasan threshold
    if ((endX - startX) <= minSize || (endY - startY) <= minSize || error <= threshold) {
        // Jika iya maka normalisasi sub blok gambar
        normalizeSubImage(im, startX, endX, startY, endY);
        return;
    }


    int midX = (startX + endX) / 2;
    int midY = (startY + endY) / 2;

    // Buat "Anak"
    divideNConquer(im, threshold, startX, midX, startY, midY, minSize);
    divideNConquer(im, threshold, midX+1, endX, startY, midY, minSize);
    divideNConquer(im, threshold, startX, midX, midY+1, endY, minSize);
    divideNConquer(im, threshold, midX+1, endX, midY+1, endY, minSize);
}
```

4. errorcalculations.h

```
#ifndef ERRORCALCULATIONS_H
#define ERRORCALCULATIONS_H

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <math.h>
#include <stdlib.h>

extern int choice;
extern int threshold;
extern int minSize;

/**
 * @brief Pilih metode perhitungan error dari 1 - 5
 */
void chooseErrorMethod();

/**
* @brief Menghitung Eror dari blok gambar berdasarkan metode yang dipilih
* @param data Pointer ke data gambar
```

```
* @param width Lebar gambar
* @param height Tinggi gambar
* @param startX Koordinat X awal blok
* @param endX Koordinat X akhir blok
* @param startY Koordinat Y awal blok
* @param endY Koordinat Y akhir blok
* @return Error value dari blok data gambar
*/
double calculateError(uint8_t *data, int width, int height, int startX, int endX, int startY, int
endY);

#endif // ERRORCALCULATIONS_H
```

5. errorcalculations.c

```
#include "header/errorcalculations.h"

int choice;
int threshold;
int minSize;

void chooseErrorMethod()
{
    while(true)
    {
        printf("\033[1;33m[Input]\033[0m Pilih metode perhitungan error :
\n");
        printf("1. Metode Variance\n");
        printf("2. Metode Mean Absoulte Deviation (MAD)\n");
        printf("3. Metode Max Pixel Differences\n");
        printf("4. Metode Entropy\n");
        printf("5. Metode Structural Similarity Index (SSI) [ NOTE : TBA
]]\n");
        printf("> ");
        scanf("%d", &choice);
        if(choice < 1 || choice > 5)
        {
            printf("\n\033[1;31m[ERROR]\033[0m Pilihan tidak valid >:(,
silahkan coba lagi.\n");
            continue;
```

```
        }
        else
        {
            printf("\033[1;33m[Input]\033[0m Masukkan Threshold : ");
            scanf("%d", &threshold);
            printf("\033[1;33m[Input]\033[0m Masukkan ukuran blok minimum :
");
            scanf("%d", &minSize);
            break;
        }
    }
}


double calculateVariance(uint8_t *data, int width, int height, int startX, int
endX, int startY, int endY) {
    double mean[3] = {0, 0, 0};
    double variance[3] = {0, 0, 0};
    int count = (endX - startX) * (endY - startY);

    if (count == 0) return 0.0;

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
                mean[c] += data[idx + c];
            }
        }
    }

    for (int c = 0; c < 3; c++) {
        mean[c] /= count;
    }

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
```

```
                double diff = data[idx + c] - mean[c];
                variance[c] += diff * diff;
            }
        }
    }

    return (variance[0] + variance[1] + variance[2]) / (3.0 * count);
}

double calculateMAD(uint8_t *data, int width, int height, int startX, int
endX, int startY, int endY) {
    double mean[3] = {0, 0, 0};
    double mad[3] = {0, 0, 0};
    int count = (endX - startX) * (endY - startY);

    if (count == 0) return 0.0;

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
                mean[c] += data[idx + c];
            }
        }
    }

    for (int c = 0; c < 3; c++) {
        mean[c] /= count;
    }

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
                mad[c] += fabs(data[idx + c] - mean[c]);
            }
        }
    }
```

```
    return (mad[0] + mad[1] + mad[2]) / (3.0 * count);
}

double calculateMaxPixelDifferences(uint8_t *data, int width, int height, int
startX, int endX, int startY, int endY)
{
    double maxDiff = 0.0;
    int count = (endX - startX) * (endY - startY);

    if (count == 0) return 0.0;

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
                double diff = fabs(data[idx + c] - data[idx + c]);
                if (diff > maxDiff) {
                    maxDiff = diff;
                }
            }
        }
    }

    return maxDiff / count;
}

double compute_entropy(int *hist, int count) {
    if (count == 0) return 0.0;

    double entropy = 0.0;
    for (int i = 0; i < 256; i++) {
        if (hist[i] > 0) {
            double p = (double)hist[i] / count;
            entropy -= p * log2(p);
        }
    }
    return entropy;
}
```

```
double calculateEntropy(uint8_t *data, int width, int height, int startX, int
endX, int startY, int endY) {
    int histogram[3][256] = {0};
    int count = (endX - startX) * (endY - startY);

    if (count == 0) return 0.0;

    for (int y = startY; y < endY; y++) {
        for (int x = startX; x < endX; x++) {
            int idx = (y * width + x) * 3;
            for (int c = 0; c < 3; c++) {
                histogram[c][data[idx + c]]++;
            }
        }
    }

    double entropy[3] = {0.0, 0.0, 0.0};
    for (int c = 0; c < 3; c++) {
        entropy[c] = compute_entropy(histogram[c], count);
    }

    return (entropy[0] + entropy[1] + entropy[2]) / 3.0;
}

double calculateSSI()
{
    return 0.0;
}

double calculateError(uint8_t *data, int width, int height, int startX, int
endX, int startY, int endY)
{
    switch (choice) {
        case 1:
            return calculateVariance(data, width, height, startX, endX,
startY, endY);
        case 2:
            return calculateMAD(data, width, height, startX, endX, startY,
endY);
```

```
        case 3:
            return calculateMaxPixelDifferences(data, width, height, startX,
endX, startY, endY);
        case 4:
            return calculateEntropy(data, width, height, startX, endX, startY,
endY);
        case 5:
            return calculateSSI(data, width, height, startX, endX, startY,
endY);
        default:
            printf("\033[1;31m[ERROR]\033[0m Pilihan tidak valid >:(, silahkan
coba lagi.\n");
            exit(1);
    }
}
```

6. image.h

```
// Header image
// !3523028 - Muhammad Aditya Rahmadeni

#ifndef IMAGE_H
#define IMAGE_H

#include <stdint.h>
#include <stdbool.h>
#include <stdio.h>
#include <stdlib.h>
#include <zlib.h>
#include <string.h>
#include <math.h>
#include <jpeglib.h>


// enum untuk mempermudah pengelompokan tipe file
typedef enum {
    IMAGE_UNKNOWN,
```

```
    IMAGE_JPEG,
    IMAGE_PNG,
    IMAGE_BMP
} ImageFormat;

extern ImageFormat format;

// Strcut dari image yang akan diproses
typedef struct Image {
    uint8_t* data;
    int width;
    int height;
} Image;


#pragma pack(push, 1)
typedef struct {
    uint16_t type;
    uint32_t size;
    uint16_t reserved1;
    uint16_t reserved2;
    uint32_t offset;
} BMPHeader;

typedef struct {
    uint32_t headerSize;
    int32_t width;
    int32_t height;
    uint16_t planes;
    uint16_t bitCount;
    uint32_t compression;
    uint32_t imageSize;
    int32_t xPixelsPerMeter;
    int32_t yPixelsPerMeter;
    uint32_t colorsUsed;
    uint32_t colorsImportant;
} BMPInfoHeader;
#pragma pack(pop)
```

```c
// untuk save png file
extern uint32_t crc_table[256];
extern int crc_table_computed;

/**
 * @brief Membaca 16 bit dari file
 *
 * @param file Pointer ke file yang ingin dibaca
 * @return uint16_t Nilai 16 bit yang dibaca dari file
 */
void extractImage(FILE* file, Image* im);

/**
 * @brief Menyimpan gambar ke file
 *
 * @param path Path file untuk menyimpan gambar
 * @param im Pointer ke struktur Image yang akan disimpan
 * @return void
 * @note Fungsi ini akan menyimpan gambar ke file dengan format yang sesuai
 * @note Format yang didukung: JPEG, PNG, BMP
 */
void saveImage(char* path, Image *im);

#endif // IMAGE_H
```

7. image.c

```c
// Implementasi dari Image
// 13523028 - Muhammad Aditya Rahmadeni

#include "header/image.h"

uint16_t read16(FILE* file) {
    uint8_t bytes[2];
    fread(bytes, 1, 2, file);
```

```
    return (bytes[0] << 8) | bytes[1];
}

uint32_t read32(FILE* file) {
    uint8_t bytes[4];
    fread(bytes, 1, 4, file);
    return (bytes[0] << 24) | (bytes[1] << 16) | (bytes[2] << 8) | bytes[3];
}

ImageFormat format;
uint32_t crc_table[256];
int crc_table_computed = 0;

ImageFormat detectImageFormat(FILE* file) {
    uint8_t signature[8];
    fread(signature, 1, 8, file);
    rewind(file);

    if (signature[0] == 0xFF && signature[1] == 0xD8) {
        return IMAGE_JPEG;
    } else if (signature[0] == 0x89 && signature[1] == 0x50 && signature[2] ==
0x4E && signature[3] == 0x47) {
        return IMAGE_PNG;
    } else if (signature[0] == 0x42 && signature[1] == 0x4D) {
        return IMAGE_BMP;
    }
    return IMAGE_UNKNOWN;
}

void extractJPEG(FILE* file, Image* im) {
    struct jpeg_decompress_struct cinfo;
    struct jpeg_error_mgr jerr;

    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_decompress(&cinfo);
    jpeg_stdio_src(&cinfo, file);
    jpeg_read_header(&cinfo, TRUE);
    jpeg_start_decompress(&cinfo);
```

```
    im->width = (uint32_t)cinfo.output_width;
    im->height = (uint32_t)cinfo.output_height;
    int channels = 3;

    printf("\033[1;32m[INFO]\033[0m Dimensi Gambar JPEG: \033[1;34m%d x
%d\033[0m\n", im->width, im->height);
      // Store channels
    im->data = (uint8_t*)malloc(im->width * im->height * channels);
    uint8_t* row_pointer[1];

    while (cinfo.output_scanline < cinfo.output_height) {
        row_pointer[0] = im->data + cinfo.output_scanline * im->width *
channels;
        jpeg_read_scanlines(&cinfo, row_pointer, 1);
    }

    jpeg_finish_decompress(&cinfo);
    jpeg_destroy_decompress(&cinfo);
    fclose(file);

    printf("\033[1;32m[INFO]\033[0m Gambar JPEG berhasil diekstrak\n\n");
}


void paethFilter(uint8_t* scanline, uint8_t* prevScanline, int stride, int
filterType) {
    for (int x = 0; x < stride; x++) {
        uint8_t a = (x >= 4) ? scanline[x - 4] : 0;
        uint8_t b = (prevScanline) ? prevScanline[x] : 0;
        uint8_t c = (prevScanline && x >= 4) ? prevScanline[x - 4] : 0;

        int p = a + b - c;
        int pa = abs(p - a);
        int pb = abs(p - b);
        int pc = abs(p - c);

        if (pa <= pb && pa <= pc) scanline[x] += a;
        else if (pb <= pc) scanline[x] += b;
        else scanline[x] += c;
```

```
    }
}

void applyPNGFilter(uint8_t* scanline, uint8_t* prevScanline, int stride, int
filterType) {
    switch (filterType) {
        case 0: // None
            break;
        case 1: // Sub
            for (int i = 3; i < stride; i++) {
                scanline[i] += scanline[i - 3];
            }
            break;
        case 2: // Up
            if (prevScanline) {
                for (int i = 0; i < stride; i++) {
                    scanline[i] += prevScanline[i];
                }
            }
            break;
        case 3: // Average
            for (int i = 0; i < stride; i++) {
                uint8_t left = (i >= 3) ? scanline[i - 3] : 0;
                uint8_t up = (prevScanline) ? prevScanline[i] : 0;
                scanline[i] += (left + up) / 2;
            }
            break;
        case 4: // Paeth
            paethFilter(scanline, prevScanline, stride, filterType);
            break;
        default:
            printf("[ERROR] Unknown PNG filter type: %d\n", filterType);
            break;
    }
}

void extractPNG(FILE* file, Image* im) {
    fseek(file, 8, SEEK_SET);
    char chunkType[5];
```

```
    uint32_t chunkSize;
    uint8_t* compressedData = NULL;
    size_t compressedSize = 0;

    while (!feof(file)) {
        chunkSize = read32(file);
        fread(chunkType, 1, 4, file);
        chunkType[4] = '\0';

        if (strcmp(chunkType, "IHDR") == 0) {
            im->width = read32(file);
            im->height = read32(file);
            fseek(file, 5, SEEK_CUR);
        } else if (strcmp(chunkType, "IDAT") == 0) {
            compressedData = (uint8_t*)realloc(compressedData, compressedSize
+ chunkSize);
            fread(compressedData + compressedSize, 1, chunkSize, file);
            compressedSize += chunkSize;
        } else if (strcmp(chunkType, "IEND") == 0) {
            break;
        } else {
            fseek(file, chunkSize, SEEK_CUR);
        }
        fseek(file, 4, SEEK_CUR);
    }
    printf("\033[1;32m[INFO]\033[0m Dimensi Gambar PNG: \033[1;34m%d x
%d\033[0m\n", im->width, im->height);

    uint8_t* decompressedData = (uint8_t*)malloc(im->width * im->height * 3 +
im->height);
    uLongf decompressedSize = im->width * im->height * 3 + im->height;
    if (uncompress(decompressedData, &decompressedSize, compressedData,
compressedSize) != Z_OK) {
        printf("\033[1;31m[ERROR]\033[0m Dekompresi PNG Gagal: Data mungkin
rusak atau format tidak didukung\n");
        free(compressedData);
        free(decompressedData);
        return;
    }
```

```
    int stride = im->width * 3;
    uint8_t* prevScanline = NULL;
    im->data = (uint8_t*)malloc(im->width * im->height * 3);
    if (!im->data) {
        printf("\033[1;31m[ERROR]\033[0m Gagal mengalokasikan memori untuk
data gambar\n");
        free(compressedData);
        free(decompressedData);
        exit(0);
    }

    for (int y = 0; y < im->height; y++) {
        int filterType = decompressedData[y * (stride + 1)];
        uint8_t* scanline = &decompressedData[y * (stride + 1) + 1];

        applyPNGFilter(scanline, prevScanline, stride, filterType);

        for (int x = 0; x < im->width; x++) {
            int baseIdx = (y * im->width + x) * 3;
            im->data[baseIdx]     = scanline[x * 3];     // R
            im->data[baseIdx + 1] = scanline[x * 3 + 1]; // G
            im->data[baseIdx + 2] = scanline[x * 3 + 2]; // B
        }

    prevScanline = scanline;
}

    free(compressedData);
    free(decompressedData);
}

void extractBMP(FILE* file, struct Image* im) {
    BMPHeader header;
    BMPInfoHeader info;

    fread(&header, sizeof(BMPHeader), 1, file);
    fread(&info, sizeof(BMPInfoHeader), 1, file);
```

```
    im->width = (uint32_t)info.width;
    im->height = (uint32_t)abs(info.height);
    printf("\033[1;32m[INFO]\033[0m Dimensi Gambar BMP: \033[1;34m%d x
%d\033[0m\n", im->width, im->height);
    if (info.bitCount != 24) {
        printf("\033[1;31m[ERROR]\033[0m Format BMP tidak didukung (hanya
24-bit RGB)\n");
        return;
    }

    fseek(file, header.offset, SEEK_SET);

    int rowSize = ((info.bitCount * im->width + 31) / 32) * 4;
    uint8_t* bmpData = (uint8_t*)malloc(rowSize * im->height);
    fread(bmpData, rowSize, im->height, file);

    im->data = (uint8_t*)malloc(im->width * im->height * 3); // Allocate
memory for RGB data

    for (int y = 0; y < im->height; y++) {
        for (int x = 0; x < im->width; x++) {
            int index = y * rowSize + x * 3;
            uint8_t b = bmpData[index];
            uint8_t g = bmpData[index + 1];
            uint8_t r = bmpData[index + 2];
            im->data[(y * im->width + x) * 3] = r;
            im->data[(y * im->width + x) * 3 + 1] = g;
            im->data[(y * im->width + x) * 3 + 2] = b;
        }
    }

    free(bmpData);
}

void extractImage(FILE* file, struct Image* im) {
    format = detectImageFormat(file);

    switch (format) {
        case IMAGE_JPEG:
```

```
            extractJPEG(file, im);
            break;
        case IMAGE_PNG:
            extractPNG(file, im);
            break;
        case IMAGE_BMP:
            extractBMP(file, im);
            break;
        default:
            printf("\033[1;31m[ERROR]\033[0m File bukan bertipe gambar
(.JPG/.JPEG, .PNG, .BMP)\n");
            break;
    }
}



void saveJPEG(char* path, Image* im) {
    struct jpeg_compress_struct cinfo;
    struct jpeg_error_mgr jerr;

    // Minta input nama file
    printf("\033[1;33m[Input]\033[0m Masukkan nama file atau path lengkap file
untuk menyimpan gambar BMP (DENGAN ekstensi) [ CATATAN : Jika hanya memberikan
nama file maka file disimpan didalam folder test]: ");
    char filename[100];
    scanf("%99s", filename); // Limit input to avoid overflow
    snprintf(path, 256, "%s%s", path, filename); // Safely concatenate strings
    FILE* file = fopen(path, "wb");
    if (!file) {
        FILE* file = fopen(filename, "wb");
        if(!file){
        printf("\033[1;31m[ERROR]\033[0m Gagal menyimpan gambar JPEG\n");
        return;
        }
    }
    cinfo.err = jpeg_std_error(&jerr);
    jpeg_create_compress(&cinfo);
    jpeg_stdio_dest(&cinfo, file);
```

```
    cinfo.image_width = im->width;
    cinfo.image_height = im->height;
    if (im->data == NULL) {
        printf("\033[1;31m[ERROR]\033[0m Data gambar tidak valid\n");
        fclose(file);
        return;
    }
    cinfo.input_components = 3;  // Assuming RGB format
    cinfo.in_color_space = JCS_RGB;
    int channels = 3;

    jpeg_set_defaults(&cinfo);

    // Set quality to 100 to avoid compression
    jpeg_set_quality(&cinfo, 100, TRUE);

    jpeg_start_compress(&cinfo, TRUE);

    uint8_t* row_pointer[1];
    while (cinfo.next_scanline < cinfo.image_height) {
        row_pointer[0] = &im->data[cinfo.next_scanline * im->width *
channels];
        jpeg_write_scanlines(&cinfo, row_pointer, 1);
    }

    jpeg_finish_compress(&cinfo);
    jpeg_destroy_compress(&cinfo);
    fclose(file);

    printf("\033[1;32m[INFO]\033[0m Gambar JPEG berhasil disimpan sebagai
%s\n\n", filename);
}


void saveBMP(char *path, Image *im) {
    printf("\033[1;33m[Input]\033[0m Masukkan nama file atau path lengkap file
untuk menyimpan gambar BMP (DENGAN ekstensi) [ CATATAN : Jika hanya memberikan
nama file maka file disimpan didalam folder test]: ");
```

```
    char filename[100];
    scanf("%99s", filename); // Limit input to avoid overflow
    snprintf(path, 256, "%s%s", path, filename); // Safely concatenate strings
    FILE* file = fopen(path, "wb");
    if (!file) {
        FILE* file = fopen(filename, "wb");
        if(!file){
            printf("\033[1;31m[ERROR]\033[0m Gagal menyimpan gambar BMP\n");
            return;
        }
    }

    BMPHeader header;
    BMPInfoHeader info;

    header.type = 0x4D42; // 'BM'
    header.size = sizeof(BMPHeader) + sizeof(BMPInfoHeader) + im->width *
im->height * 3;
    header.reserved1 = 0;
    header.reserved2 = 0;
    header.offset = sizeof(BMPHeader) + sizeof(BMPInfoHeader);

    info.headerSize = sizeof(BMPInfoHeader);
    info.width = im->width;
    info.height = im->height;
    info.planes = 1;
    info.bitCount = 24; // RGB
    info.compression = 0; // BI_RGB
    info.imageSize = im->width * im->height * 3;
    info.xPixelsPerMeter = 0;
    info.yPixelsPerMeter = 0;
    info.colorsUsed = 0;
    info.colorsImportant = 0;

    fwrite(&header, sizeof(BMPHeader), 1, file);
    fwrite(&info, sizeof(BMPInfoHeader), 1, file);

    for (int y = 0; y < im->height; y++) {
        for (int x = 0; x < im->width; x++) {
```

```
            int index = (y * im->width + x) * 3;
            uint8_t bgr[3] = {im->data[index + 2], im->data[index + 1],
im->data[index]}; // BMP uses BGR format
            fwrite(bgr, sizeof(uint8_t), 3, file);
        }
        // Padding for each row to be a multiple of 4 bytes
        uint8_t padding[3] = {0, 0, 0};
        fwrite(padding, sizeof(uint8_t), (4 - (im->width * 3) % 4) % 4, file);
    }

    fclose(file);
    printf("\033[1;32m[INFO]\033[0m Gambar BMP berhasil disimpan sebagai
%s\n\n", path);
}

void make_crc_table() {
    for (uint32_t n = 0; n < 256; n++) {
        uint32_t c = n;
        for (int k = 0; k < 8; k++) {
            c = c & 1 ? 0xedb88320L ^ (c >> 1) : c >> 1;
        }
        crc_table[n] = c;
    }
    crc_table_computed = 1;
}

uint32_t update_crc(uint32_t crc, uint8_t *buf, int len) {
    uint32_t c = crc;
    if (!crc_table_computed)
        make_crc_table();
    for (int n = 0; n < len; n++) {
        c = crc_table[(c ^ buf[n]) & 0xff] ^ (c >> 8);
    }
    return c;
}

uint32_t crc(uint8_t *buf, int len) {
    return update_crc(0xffffffffL, buf, len) ^ 0xffffffffL;
}
```

```
void write32(FILE* f, uint32_t val) {
    fputc((val >> 24) & 0xFF, f);
    fputc((val >> 16) & 0xFF, f);
    fputc((val >> 8) & 0xFF, f);
    fputc(val & 0xFF, f);
}

void writeChunk(FILE* f, const char* type, uint8_t* data, uint32_t length) {
    write32(f, length);
    fwrite(type, 1, 4, f);
    if (length > 0) fwrite(data, 1, length, f);

    uint32_t crc_val = crc((uint8_t*)type, 4);
    if (length > 0) crc_val = update_crc(crc_val, data, length);
    write32(f, crc_val);
}


void savePNG(char* path, Image* im) {
    printf("\033[1;33m[Input]\033[0m Masukkan nama file atau path lengkap file
untuk menyimpan gambar PNG (DENGAN ekstensi) [ CATATAN : Jika hanya memberikan
nama file maka file disimpan didalam folder test]: ");
    char filename[100];
    scanf("%99s", filename); // Limit input to avoid overflow
    snprintf(path, 256, "%s%s", path, filename); // Safely concatenate strings
    FILE* f = fopen(path, "wb");
    if (!f) {
        FILE* f = fopen(filename, "wb");
        if(!f){
            printf("\033[1;31m[ERROR]\033[0m Gagal menyimpan gambar BMP\n");
            return;
        }
    }

    // PNG signature
    uint8_t sig[8] = {137, 80, 78, 71, 13, 10, 26, 10};
    fwrite(sig, 1, 8, f);
```

```
    // IHDR
    uint8_t ihdr[13];
    ihdr[0] = (im->width >> 24) & 0xFF;
    ihdr[1] = (im->width >> 16) & 0xFF;
    ihdr[2] = (im->width >> 8) & 0xFF;
    ihdr[3] = im->width & 0xFF;

    ihdr[4] = (im->height >> 24) & 0xFF;
    ihdr[5] = (im->height >> 16) & 0xFF;
    ihdr[6] = (im->height >> 8) & 0xFF;
    ihdr[7] = im->height & 0xFF;

    ihdr[8] = 8;  // Bit depth
    ihdr[9] = 2;  // Color type: RGB
    ihdr[10] = 0; // Compression
    ihdr[11] = 0; // Filter
    ihdr[12] = 0; // Interlace

    writeChunk(f, "IHDR", ihdr, 13);

    size_t rawSize = (im->width * 3 + 1) * im->height;
    uint8_t* rawData = (uint8_t*)malloc(rawSize);
    for (int y = 0; y < im->height; y++) {
        rawData[y * (im->width * 3 + 1)] = 0; // Filter type 0
        memcpy(&rawData[y * (im->width * 3 + 1) + 1],
                &im->data[y * im->width * 3],
                im->width * 3);
    }

    // Compress with zlib
    uLongf compSize = compressBound(rawSize);
    uint8_t* compData = (uint8_t*)malloc(compSize);
    if (compress2(compData, &compSize, rawData, rawSize, Z_BEST_COMPRESSION)
!= Z_OK) {
        printf("\033[1;31m[ERROR]\033[0m PNG compression failed.\n");
        free(rawData);
        free(compData);
        fclose(f);
        return;
```

```
    }

    writeChunk(f, "IDAT", compData, compSize);

    // Write IEND
    writeChunk(f, "IEND", NULL, 0);

    free(rawData);
    free(compData);
    fclose(f);

    printf("\033[1;32m[INFO]\033[0m Gambar PNG berhasil disimpan sebagai
%s\n\n", path);
}

void saveImage(char* path, Image *im) {
    switch (format) {
        case IMAGE_JPEG:
            saveJPEG(path, im);
            break;
        case IMAGE_PNG:
            savePNG(path, im);
            break;
        case IMAGE_BMP:
            saveBMP(path, im);
            break;
        default:
            printf("\033[1;31m[ERROR]\033[0m Format gambar tidak didukung
untuk penyimpanan\n");
            break;
    }
}
```

BAB IV
Hasil

1. Test Case 1

Input	Output

2. Test Case 2

Input	Output

3. Test Case 3 (bmp file)

Input	Output


```
./bat1.sh
[Input] Masukkan nama file gambar yang ingin diproses ( CATATAN : PASTIKAN FILE ADA DALAM FOLDER TEST ATAU PATH LENGKAP FILE ADA )
> test3.bmp

[Success] File ditemukan !

[INFO] Dimensi Gambar BMP: 5921 x 2262
[Input] Pilih metode perhitungan error :
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Differences
4. Metode Entropy
5. Metode Structural Similarity Index (SSI) [ NOTE : TBA ]]

[Input] Masukkan Threshold : 20
[Input] Masukkan ukuran blok minimum : 8
[INFO] Gambar akan mulai diproses...

[INFO] Gambar berhasil diproses dalam waktu 1.37 detik
[INFO] Kedalaman pohon: 10
[INFO] Banyak simpul anak pohon: 1320
[Input] Masukkan nama file atau path lengkap file untuk menyimpan gambar BMP (DENGAN ekstensi) [ CATATAN : Jika hanya memberikan nama file maka file disimpan didalam folder test]: test3.bmp

[INFO] Gambar BMP berhasil disimpan sebagai test3.bmp
```

4. Test Case 4

Input	Output

```
./bat1.sh
[Input] Masukkan nama file gambar yang ingin diproses ( CATATAN : PASTIKAN FILE ADA DALAM FOLDER TEST ATAU PATH LENGKAP FILE ADA )
> test2.jpg
[INFO] File pada test2.jpg ditemukan . /test/test2.jpg tidak ditemukan, cobakan cara lagi
[Input] Masukkan nama file gambar yang ingin diproses ( CATATAN : PASTIKAN FILE ADA DALAM FOLDER TEST ATAU PATH LENGKAP FILE ADA )
> test2.jpg

[Success] File ditemukan !

[INFO] Dimensi Gambar JPEG: 3068 x 2168
[INFO] Gambar JPEG berhasil diekstrak

[Input] Pilih metode perhitungan error :
1. Metode Variance
2. Metode Mean Absolute Deviation (MAD)
3. Metode Max Pixel Differences
4. Metode Entropy
5. Metode Structural Similarity Index (SSI) [ NOTE : TBA ]]

[Input] Masukkan Threshold : 200
[Input] Masukkan ukuran blok minimum : 4
[INFO] Gambar akan mulai diproses...

[INFO] Gambar berhasil diproses dalam waktu 1.15 detik
[INFO] Kedalaman pohon: 10
[INFO] Banyak simpul anak pohon: 2707
[Input] Masukkan nama file atau path lengkap file untuk menyimpan gambar JPG (DENGAN ekstensi) [ CATATAN : Jika hanya memberikan nama file maka file disimpan didalam folder test]: test4.jp

[INFO] Gambar JPEG berhasil disimpan sebagai 4.jpg
```

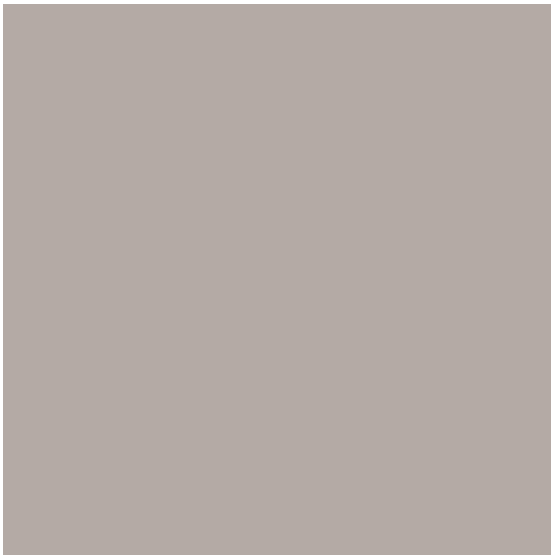
5. Test Case 5

Input	Output

```
❯ ./build.sh
[Input] Masukkan nama file gambar yang ingin diproses ( CATATAN : PASTIKAN FILE ADA DALAM FO
LDER TEST ATAU PATH LENGKAP FILE ADA )
> test4.jpg

[Success] File ditemukan !

[INFO] Dimensi Gambar JPEG: 283 x 200
[INFO] Gambar JPEG berhasil diekstrak

[Input] Pilih metode perhitungan error :
1. Metode Variance
2. Metode Mean Absoulte Deviation (MAD)
3. Metode Max Pixel Differences
4. Metode Entropy
5. Metode Structural Similarity Index (SSI) [ NOTE : TBA ]]
> 1
[Input] Masukkan Threshold : 1000
[Input] Masukkan ukuran blok minimum : 1000
[INFO] Gambar akan mulai diproses...

[INFO] Gambar berhasil diproses dalam waktu 0.01 detik
[INFO] Kedalaman pohon: 0
[INFO] Banyak simpul pada pohon: 1
[Input] Masukkan nama file atau path lengkap file untuk menyimpan gambar JPG (DENGAN ekstens
i) [ CATATAN : Jika hanya memberikan nama file maka file disimpan didalam folder test]: outp
ut.jpg
[INFO] Gambar JPEG berhasil disimpan sebagai output.jpg
```

6. Test Case 6

Input	Output

7. Test Case 7

Input	Output

BAB V
ANALISIS ALGORITMA

Algoritma *Divide and Conquer* berbasis Quadtree telah diimplementasikan dan diuji terhadap berbagai gambar berwarna dengan dimensi yang bervariasi. Berdasarkan hasil percobaan, algoritma mampu secara efektif memecah gambar menjadi blok-blok kecil dan melakukan normalisasi warna terhadap bagian-bagian gambar yang memiliki tingkat variasi

warna rendah (homogen). Hal ini terbukti dari banyaknya blok yang tidak lagi diproses lebih lanjut setelah memenuhi kriteria threshold error yang ditentukan.

Proses pembagian blok dilakukan secara rekursif hingga memenuhi salah satu dari dua kondisi penghentian, yaitu ukuran blok lebih kecil dari batas minimum (*minSize*) atau nilai error berada di bawah nilai ambang batas (*threshold*). Blok-blok homogen akan mengalami proses normalisasi, sementara blok yang heterogen akan terus dibagi. Strategi ini membuat algoritma bersifat adaptif terhadap struktur visual gambar: bagian gambar yang kompleks akan memiliki lebih banyak pembagian, sedangkan bagian yang datar atau berwarna seragam akan disederhanakan.

Dari sisi performa, waktu eksekusi algoritma sangat bergantung pada parameter *threshold* dan *minSize*. Nilai threshold yang rendah akan menyebabkan pembagian lebih dalam, sehingga menambah beban komputasi, sedangkan nilai threshold yang tinggi akan mempercepat proses namun berpotensi menurunkan kualitas visual hasil akhir.

- **Kompleksitas Waktu (Time Complexity):**
  Setiap piksel hanya diproses satu kali, baik saat perhitungan error maupun normalisasi. Dengan total $W \times H$ piksel, maka:

  $$T(W, H) = O(W \times H)$$

- **Kompleksitas Ruang (Space Complexity):**

  Karena algoritma bersifat rekursif dan tidak menyimpan struktur pohon secara eksplisit, penggunaan memori bergantung pada kedalaman maksimum rekursi, maka

  $$O(\log_{10}(W \times H))$$

LAMPIRAN

Pranala Github : https://github.com/Kurosue/Tucil2_13523028

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	✓	