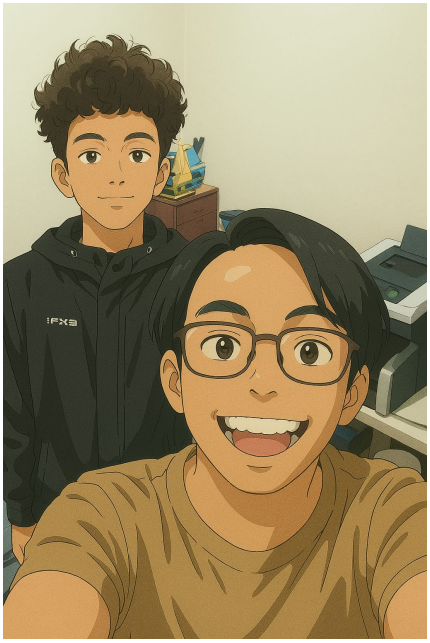
Laporan Tugas Kecil 2

Kompresi Gambar Dengan Metode Quadtree

Disusun untuk memenuhi tugas mata kuliah IF2211 Strategi Algoritma pada Semester 2 (Genap) Tahun Akademik 2024/2025



Disusun oleh:

Razi Rachman Widyadhana	13523004
Mochammad Fariz Rifqi Rizqulloh	13523069

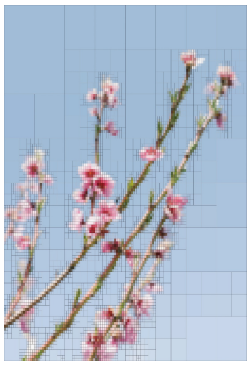
PROGRAM STUDI TEKNIK INFORMATIKA
SEKOLAH TEKNIK ELEKTRO DAN INFORMATIKA
INSTITUT TEKNOLOGI BANDUNG
JL. GANESA 10, BANDUNG 40132

2025

Daftar Isi

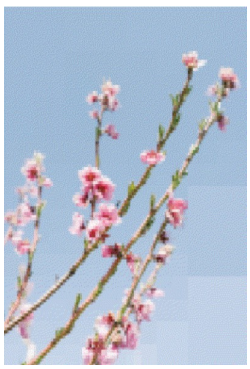
Bab I
Deskripsi Masalah

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.



Gambar 1: Quadtree dalam Kompresi Gambar (Sumber: Medium - @tannerwyork)

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.



Gambar 2: GIF Proses Pembentukan Quadtree dalam Kompresi Gambar (Lihat: Medium)

Bab II
Algoritma Divide and Conquer pada Kompresi Gambar

2.1. Algoritma *Divde and Conquer*

Algoritma *Divide and Conquer* secara rekursif memecah persoalan menjadi dua atau lebih upa-persoalan dengan tipe yang sama atau mirip dari persoalan semula sehingga persoalan tersebut menjadi cukup sederhana untuk diselesaikan secara langsung (menuju kasus basis upa-persoalan). Solusi dari upa-persoalan tersebut kemudian digabungkan untuk memberikan solusi bagi persoalan semula.

Algoritma *Divide and Conquer* dibagi menjadi tiga tahap utama:

1. **Divide:**
   Membagi persoalan menjadi beberapa upa-persoalan yang memiliki kemiripan dengan persoalan semula namun berukuran lebih kecil (idealnya setiap upa-persoalan berukuran hampir sama),

2. **Conquer:**
   Menyelesaikan (*solve*) masing-masing upa-persoalan (secara langsung jika sudah berukuran kecil atau secara rekursif jika masih berukuran besar).

3. **Combine:**
   Menggabungkan solusi masing-masing upa-persoalan sehingga membentuk solusi persoalan semula.

Algoritma ini serupa dengan algoritma *Decrease and Conquer*, hanya saja *Divide and Conquer* lebih menekankan optimalisasi kompleksitas dibanding pengurangan ukuran persoalannya. Karena *Divide and Conquer* menyelesaikan upa-persoalan secara rekursif, setiap upa-persoalan harus lebih kecil dari persoalan semulanya, dan harus ada kasus basis untuk upa-persoalan.

Algoritma ini sangat berguna saat membagi sebuah persoalan menjadi beberapa upa-persoalan yang independen. Jika upa-persoalan *overlapping* satu sama lain, maka digunakan *Dynamic Programming*. Selain itu, *Divide and Conquer* sering digunakan sebagai dasar dari algoritma yang efisien untuk banyak persoalan, seperti pengurutan (*quicksort* dan *mergesort*), mengalikan bilangan besar (algoritma Karatsuba), dan menemukan pasangan titik terdekat.

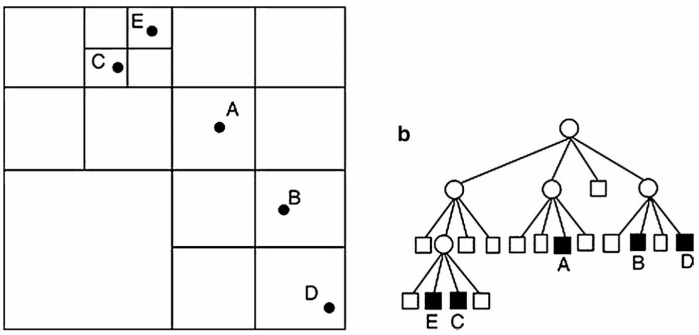
2.2. Abstraksi Permasalahan

Seperti yang telah didefinisikan sebelumnya, algoritma *Divide and Conquer* akan membagi persoalan yang rumit menjadi persoalan-persoalan (upa-persoalan) yang lebih sederhana untuk diselesaikan secara langsung.

Ingat kembali bahwa pada Tugas Kecil 2 ini menggunakan struktur data Quadtree dan terdapat parameter-parameter yang akan digunakan untuk menentukan kompresi gambar yang diinginkan. Dengan meninjau masing-masing parameter terlebih dahulu, akan memudahkan proses abstraksi untuk algoritma *Divide and Conquer* ini.

2.2.1 Quadtree

Quadtree dalam kompresi gambar bekerja dengan membagi gambar secara rekursif menjadi empat upa-ruang dengan masing-masing menyimpan *error* dan warna RGB rata-rata yang menentukan warna tersebut untuk upa-ruangnya. *Threshold* pun ditetapkan berdasarkan *error* tersebut dan membantu pohon menentukan apakah sebuah simpul harus dipecah lebih lanjut atau tidak.



Gambar 3: Struktur Data Quadtree dalam Kompresi Gambar

Pada konteks kompresi menggunakan algoritma Quadtree, Setiap simpul pada pohon Quadtree mewakili satu bagian area (blok) gambar, dan penyimpanan nilai rata-rata warna serta tingkat error memungkinkan kompresi dilakukan secara adaptif. Area gambar yang homogen tidak perlu dibagi lebih lanjut dan cukup disimpan sebagai satu simpul dengan warna tunggal, sementara area yang lebih kompleks akan dianalisis lebih detail. Dengan cara ini, Quadtree tidak hanya merepresentasikan struktur spasial dari gambar, tetapi juga mendekati prinsip efisiensi data: mempertahankan detail hanya di area yang penting, dan menyederhanakan area yang tidak mengandung banyak informasi visual. Pendekatan ini membuat Quadtree menjadi teknik yang efisien untuk kompresi karena mampu mengurangi ukuran data tanpa kehilangan detail penting secara berlebihan.

2.2.2 Paramter-Parameter Kompresi Gambar

Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

a. 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.

- Variance

$$\sigma_c^2 = \frac{1}{N}\sum_{i\,=\,1}^{N}(P_{i,\,c} - \mu_c)^2 \quad \text{dan} \quad \sigma_{RGB}^2 = \frac{\sigma_R^2 + \sigma_G^2 + \sigma_B^2}{3}$$

Dalam hal ini:

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

Akan tetapi, rumus di atas masih dapat disederhanakan kembali menjadi

$$\begin{aligned}
\sigma_c^2 &= \frac{1}{N}\sum_{i\,=\,1}^{N}\left[(P_{i,\,c})^2 - 2P_{i,\,c}\,\mu_c + \mu_c^2\,\right] \\
&= \frac{\sum(P_{i,\,c})^2}{N} - \frac{2\mu_c\,\sum P_{i,\,c}}{N} + \frac{\sum \mu_c^2}{N} \\
&= \frac{\sum(P_{i,\,c})^2}{N} - 2\mu_c \cdot \mu_c + \frac{N \cdot \mu_c^2}{N} \\
&= \frac{\sum(P_{i,\,c})^2}{N} - 2\mu_c^2 + \mu_c^2 \\
&= \frac{\sum(P_{i,\,c})^2}{N} - \mu_c^2
\end{aligned}$$

Dari sudut pandang kompleksitas waktu, yang sebelumnya memerlukan kompleksitas waktu eksekusi $O(2 \times w \times h)$ karena memerlukan *loop* sekali untuk menghitung rataan dan sekali lagi untuk menghitung variansi, berubah menjadi memiliki kompleksitas waktu eksekusi $O(w \times h)$ dengan menyimpan jumlah nilai dan jumlah kuadrat nilai secara bersamaan. Akibatnya, hal ini memberikan efisiensi komputasi yang signifikan.

Lebih lanjut, dapat dimanfaatkan *Prefix Sum 2D* yang menyimpan akumulasi nilai piksel dan kuadratnya dalam matriks 2D untuk masing-masing kanal warna (R, G, dan B) sehingga memungkinkan pengambilan nilai total, rata-rata, dan variansi sebuah blok dilakukan dalam waktu konstan. Implementasi ini mengurangi kompleksitas waktu per blok dari $O(w \times h)$ menjadi $O(1)$ sehingga sangat efektif ketika harus menghitung nilai *error* untuk banyak blok secara berulang dalam proses traversal QuadTree.

- **Mean Absolute Deviation (MAD)**

$$MAD_c = \frac{1}{N}\sum_{i\,=\,1}^{N}|P_{i,\,c} - \mu_c|$$

dan

$$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$$

Dalam hal ini:

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
μ_c	=	Nilai rata-rata tiap piksel dalam satu blok
N	=	Banyaknya piksel dalam satu blok

- Max Pixel Difference

$$D_c = \max\left(P_{i,\ c}\right) - \min\left(P_{i,\ c}\right) \quad \text{dan} \quad D_{RGB} = \frac{D_R + D_G + D_B}{3}$$

Dalam hal ini:

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 kanal warna c

- Entropy

$$H_c = -\sum_{i\ =\ 1}^{N} P_c(i)\ \log_2\left(P_c(i)\right) \quad \text{dan} \quad H_{RGB} = \frac{H_R + H_G + H_B}{3}$$

Dalam hal ini:

H_c	=	Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
$P_{i,c}$	=	Nilai piksel pada posisi i untuk kanal warna c

- [BONUS] Structural Similarity Index (SSIM)

Nilai SSIM yang dibandingkan adalah antara blok gambar sebelum dan sesudah dikompresi. Asumsikan gambar yang akan diuji adalah 24-bit RGB dengan 8-bit per kanal.

$$SSIM_c\ (x, y) = \frac{(2\ \mu_{x,c}\ \mu_{y,c} + C_1)\,(2\ \sigma_{xy,c} + C_2)}{\left(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1\right)\left(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2\right)}$$

dan

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

Dalam hal ini:

$SSIM_c(x, y)$	=	Nilai SSIM untuk kanal warna c (R, G, B) pada citra x dan y
$\sigma_{xy,c}$	=	Kovarians antara nilai piksel citra x dan y pada kanal warna c
C_1, C_2	=	Konstanta kecil untuk menjaga stabilitas ketika penyebut mendekati nol

Dalam kasus ini, y merupakan blok seragam dengan nilai rata-rata yang sama dengan x ($\mu_y = \mu_x$ dan $\sigma_y = 0$), akibatnya juga $\sigma_{xy} = 0$. Maka persamaan ini dapat disederhanakan menjadi

$$\begin{aligned} SSIM_c(x,y) &= \frac{(2\ \mu_{x,c}\ \mu_{x,c} + C_1)\,(2 \cdot 0 + C_2)}{\left(\mu_{x,c}^2 + \mu_{x,c}^2 + C_1\right)\left(\sigma_{x,c}^2 + 0 + C_2\right)} \\ &= \frac{(2\ \mu_c^2 + C_1)\,(C_2)}{(2\mu_c^2 + C_1)\,(\sigma_c^2 + C_2)} \\ &= \frac{C_2}{\sigma_c^2 + C^2} \end{aligned}$$

Dalam hal ini:

C_2 = $(0.03 \cdot 255)^2$
σ_c = Standar deviasi nilai piksel pada kanal warna c

Penyederhanaan ini sangat mengoptimalisasi persamaan yang sebelumnya dengan mengurangi variabel-variabel yang ada. Akibatnya, kalkulasi yang diperlukan menjadi lebih sedikit dan mengurangi waktu eksekusi dalam perhitungan *error*.

Seperti sebelumnya pada metode variansi, dapat digunakan juga *Prefix Sum 2D* yang sangat efektif ketika harus menghitung nilai *error* untuk banyak blok secara berulang dalam proses traversal QuadTree.

b. Threshold (Ambang Batas)

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

c. 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.

d. [BONUS] Compression Percentage (Persentase Kompresi)

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

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

2.2.3 *Algoritma Penyelesaian Masalah*

Dari proses abstraksi yang telah dipaparkan, diperoleh tahapan untuk melakukan kompresi gambar dengan metode Quadtree secara algoritmik. Adapun algoritma *Divide and Conquer* yang digunakan adalah sebagai berikut.

1. Baca citra dari masukan dan representasikan sebagai matriks 2D piksel dengan dimensi sesuai ukuran gambar, serta siapkan parameter seperti nilai *threshold*, ukuran minimum blok, dan metode perhitungan *error*.
2. Bangun simpul akar dari Quadtree yang mewakili keseluruhan gambar, dengan mencatat koordinat awal, ukuran blok, dan menghitung nilai rata-rata warna serta nilai *error* pada blok tersebut.
3. Evaluasi blok berdasarkan nilai *error* yang dihitung; jika nilai *error* melebihi *threshold* dan ukuran blok masih di atas batas minimum, maka blok tersebut perlu dibagi menjadi empat sub-blok kuadran.
4. Untuk setiap sub-blok yang terbentuk, ulangi proses pada poin [3] secara rekursif, sehingga pohon akan terus bertambah ke bawah jika masih ditemukan blok-blok yang belum seragam.
5. Jika sebuah blok memenuhi kriteria keseragaman atau telah mencapai ukuran minimum, maka blok tersebut dianggap sebagai simpul daun, dan warna rata-rata dari blok digunakan untuk menggantikan seluruh piksel dalam wilayah tersebut.
6. Setelah seluruh proses rekursi selesai, hasil akhirnya berupa gambar baru yang dibentuk ulang dari struktur pohon Quadtree, di mana setiap blok yang tidak dibagi lebih lanjut akan direpresentasikan dengan warna rata-ratanya.

Bab III
Implementasi Program dengan C++

Algoritma yang telah dijelaskan pada Bab 2 diimplementasikan dengan bahasa pemrograman C++. Untuk menyesuaikan proses perancangan program yang *intended* pada C++, digunakan pendekatan *object-oriented programming*, di mana setiap komponen utama program akan dienkapulasi oleh *Object*. Untuk program ini, sebenarnya terdapat beberapa kelas. Akan tetapi, tiga kelas utama yang berperan sebagai *core logic* program ini adalah `QuadTree`, `QuadTreeNode`, dan `ErrorMethod`.

Selain dari 3 kelas yang digunakan, program kami juga menggunakan *external library* untuk membantu implementasi, yaitu library `STBImage` yang digunakan untuk membantu program dalam menerima masukan gambar, memproses gambar, serta menulis gambar ke dalam file. Kemudian ada juga library `gif` yang digunakan untuk memproses frame-frame yang telah terbentuk menjadi suatu GIF (Graphics Interchange Format), serta menulisnya ke dalam file system.

3.1. Kelas `QuadTree`

```
#ifndef QUADTREE_HPP
#define QUADTREE_HPP

#include <queue>
#include <time.h>
#include "QuadTreeNode.hpp"

// Global variables for image data
extern unsigned char* currImgData;
extern unsigned char* initImgData;
extern unsigned char* tempImgData;
extern int imgWidth, imgHeight, imgChannels;

class QuadTree {

    private:
        int mode, minBlock;
        double threshold, targetPercentage;
        string inputPath, inputExtension, outputPath, gifPath;

        bool lastImg;
        QuadTreeNode root;
        GifWriter g;
        uint8_t* data;

        int quadtreeDepth;
        int quadtreeNode;
```

```
int initialSize;
int finalSize;
double compressionPercentage;

time_t startTime, endTime;

void writeCurrImageToGif() {
    for (int i = 0; i < imgWidth; i++) {
        for (int j = 0; j < imgHeight; j++) {
            int idx = (j * imgWidth + i) * imgChannels;
            int outIdx = (j * imgWidth + i) * 4;

            data[outIdx + 0] = currImgData[idx + 0];
            data[outIdx + 1] = currImgData[idx + 1];
            data[outIdx + 2] = currImgData[idx + 2];

            if (imgChannels == 4) {
                data[outIdx + 3] = currImgData[idx + 3];
            }
            else {
                data[outIdx + 3] = 255;
            }
        }
    }

    GifWriteFrame(&g, data, imgWidth, imgHeight, 100);
}

void writeTempImageToGif() {
    for (int i = 0; i < imgWidth; i++) {
        for (int j = 0; j < imgHeight; j++) {
            int idx = (j * imgWidth + i) * imgChannels;
            int outIdx = (j * imgWidth + i) * 4;

            data[outIdx + 0] = tempImgData[idx + 0];
            data[outIdx + 1] = tempImgData[idx + 1];
            data[outIdx + 2] = tempImgData[idx + 2];

            if (imgChannels == 4) {
                data[outIdx + 3] = currImgData[idx + 3];
            }
            else {
                data[outIdx + 3] = 255;
            }
        }
    }

    GifWriteFrame(&g, data, imgWidth, imgHeight, 100);
}
```

```
        void writeCurrImage(string path) const {

            if (!path.empty()) {
                if (inputExtension == "png") {
                    stbi_write_png(path.c_str(), imgWidth,
                    ↪  imgHeight, imgChannels, currImgData,
                    ↪  imgWidth * imgChannels);
                }
                else {
                    stbi_write_jpg(path.c_str(), imgWidth,
                    ↪  imgHeight, imgChannels, currImgData, 90);
                }
            }
        }

        void writeTempImage(string path) const {

            if (!path.empty()) {
                if (inputExtension == "png") {
                    stbi_write_png(path.c_str(), imgWidth,
                    ↪  imgHeight, 4, tempImgData, imgWidth *
                    ↪  imgChannels);
                }
                else {
                    stbi_write_jpg(path.c_str(), imgWidth,
                    ↪  imgHeight, 3, tempImgData, 90);
                }
            }
        }

    public:
        QuadTree(string inputPath,
                 int mode,
                 double threshold,
                 int minBlock,
                 double targetPercentage,
                 string outputPath,
                 string gifPath,
                 string inputExtension) {

            this -> inputPath = inputPath;
            this -> mode = mode;
            this -> threshold = threshold;
            this -> minBlock = minBlock;
            this -> targetPercentage = targetPercentage;
            this -> outputPath = outputPath;
            this -> gifPath = gifPath;
            this -> inputExtension = inputExtension;
            this -> root = QuadTreeNode(0, 0, 0, imgWidth,
            ↪  imgHeight, mode);
```

```
        if (targetPercentage == 0) lastImg = true;
        else lastImg = false;

        char* gifPathCopy = new char[gifPath.length() + 1];
        strcpy(gifPathCopy, gifPath.c_str());
        GifBegin(&g, gifPathCopy, imgWidth, imgHeight, 50);

        this -> data = (uint8_t*) malloc (imgWidth * imgHeight *
        ↪  4);
        this -> initialSize = Image::getOriginalSize(inputPath);
        this -> startTime = clock();
        this -> quadtreeNode = 0;
    }

    ~QuadTree() {
        if (data != nullptr) {
            free(data);
            data = nullptr;
        }
    }

    void performQuadTree() {
        queue<QuadTreeNode> q;
        q.push(root);
        int curMaxStep = 0;
        memcpy(tempImgData, currImgData, imgWidth * imgHeight *
        ↪  imgChannels);

        while (!q.empty()) {
            QuadTreeNode node = q.front();
            q.pop();

            if (lastImg) quadtreeNode++;

            int step = node.getStep();
            int X = node.getX();
            int Y = node.getY();
            int width = node.getWidth();
            int height = node.getHeight();
            if (step < curMaxStep) continue;

            if (step > curMaxStep && lastImg) {
                quadtreeDepth = step;
                curMaxStep = step;
                writeTempImageToGif();
                memcpy(tempImgData, currImgData, width * height
                ↪  * imgChannels);
            }
```

```
            if (width == 0 || height == 0 || ((long
            ↪  long)node.getWidth() * (long
            ↪  long)node.getHeight()) < minBlock ||
            ↪  node.getError() <= threshold) {
                node.fillCurrRectangle();
                if (lastImg) node.fillTempRectangle();
                continue;
            }
            else {
                if (lastImg) node.fillTempRectangle();
                q.push(QuadTreeNode(step + 1, X, Y, width/2,
                ↪  height/2, mode));
                q.push(QuadTreeNode(step + 1, X + height/2, Y,
                ↪  width/2, height - height/2, mode));
                q.push(QuadTreeNode(step + 1, X, Y + width/2,
                ↪  width - width/2, height/2, mode));
                q.push(QuadTreeNode(step + 1, X + height/2, Y +
                ↪  width/2, width - width/2, height - height/2,
                ↪  mode));
            }
        }

        if (lastImg) {
            writeCurrImageToGif();
            writeCurrImage(outputPath);

            endTime = clock();
            finalSize = Image::getEncodedSize(currImgData,
            ↪  imgWidth, imgHeight, inputExtension,
            ↪  imgChannels);
            compressionPercentage = ((double)(initialSize -
            ↪  finalSize) / initialSize) * 100.0;

            GifEnd(&g);
            free(data);
        }
    }

    void performBinserQuadTree(double ratio) {
        ErrorMethod* errorMethod = nullptr;
        double lowerThreshold = 0.0, upperThreshold = 0.0;

        switch(mode) {
            case 1:
                errorMethod = new Variance();
                break;
            case 2:
                errorMethod = new MeanAbsoluteDeviation();
                break;
```

```
            case 3:
                errorMethod = new MaxPixelDifference();
                break;
            case 4:
                errorMethod = new Entropy();
                break;
            case 5:
                errorMethod = new SSIM();
                break;
            default:
                errorMethod = new Variance();
                break;
        }

        if (errorMethod)
        {
            lowerThreshold = errorMethod->getLowerThreshold();
            upperThreshold = errorMethod->getUpperThreshold();
            delete errorMethod;
        }

        size_t initImageSize =
        ↪  Image::getOriginalSize(inputPath);
        size_t targetImageSize = initImageSize - (initImageSize
        ↪  * ratio);

        double l = lowerThreshold, r = upperThreshold;
        double bestThreshold = -1;
        lastImg = false;

        for (int i = 1; i <= 13; i++) {
            double mid = (l + r) / 2;
            threshold = mid;
            performQuadTree();

            size_t currImgSize =
            ↪  Image::getEncodedSize(currImgData, imgWidth,
            ↪  imgHeight, inputExtension, imgChannels);

            if (currImgSize <= targetImageSize) {
                bestThreshold = mid;
                r = mid;
            }
            else l = mid;

            memcpy(currImgData, initImgData, imgWidth *
            ↪  imgHeight * imgChannels);
        }

        if (bestThreshold == -1) bestThreshold = threshold;
```

```
            lastImg = true;
            threshold = bestThreshold;
            performQuadTree();

            size_t currImgSize = Image::getEncodedSize(currImgData,
            ↪  imgWidth, imgHeight, inputExtension, imgChannels);
        }

        int getQuadtreeDepth() const {return quadtreeDepth;}
        int getQuadtreeNode() const {return quadtreeNode;}
        int getInitialSize() const {return initialSize;}
        int getFinalSize() const {return finalSize;}

        double getCompressionPercentage() const {
            return round(compressionPercentage * 100) / 100.0;
        }
        int getExecutionTime() const {
            return (endTime - startTime) / (CLOCKS_PER_SEC / 1000);
        }
};

#endif // QUADTREE_HPP
```

Kelas `Quadtree` berperan sebagai *Model* dalam program ini, yaitu kelas yang berisikan logika-logika utama, seperti logika utama algoritma kompresi quadtree, *binary search*, dan lain sebagainya. Dalam Kelas `Quadtree`, terdapat konstruktor untuk dirinya, yaitu *method* `QuadTree(string inputPath, int mode, double threshold, int minBlock, double targetPercentage, string outputPath, string gifPath)` yang menyimpan *parameter-parameter* yang diberikan sebagai atribut dari kelas tersebut. Selain itu, konstruktor tersebut akan melakukan pembuatan file `GIF` yang akan digunakan untuk menampilkan proses pembentukan *quadtree* serta melakukan inisialisasi pada atribut-atribut yang tidak bergantung pada masukan user, seperti atribut `quadtreeNode` yang merupakan variabel untuk menghitung ada berapa banyak simpul pada *quadtree*.

Pada kelas `Quadtree`, terdapat beberapa method *private* yang berperan sebagai kelas pembantu dalam proses implementasi kompresi citra `Quadtree`, seperti `void writeCurrImageToGif()`, yang berfungsi untuk menulis frame saat ini menuju GIF, serta `void writeCurrImage(string path)` yang berfungsi untuk menulis frame menuju path yang telah ditentukan. Ada juga fungsi-fungsi public *getter* yang digunakan untuk mendapatkan informasi mengenai proses quadtree, seperti `getQuadtreeDepth()`, digunakan untuk mendapatkan kedalaman dari proses quadtree. Kemudian `int getQuadtreeNode()`, digunakan untuk mendapatkan jumlah simpul dari proses quadtree. Kemudian terdapat `double getCompressionPercentage()` yang digunakan untuk mendapatkan persentase kompresi dari proses quadtree. Kemudian terdapat `int getExecutionTime()`, yang digunakan untuk mendapatkan lama eksekusi program quadtree. Dan terakhir, terdapat fungsi `getInitialSize` dan `getFinalSize`.

3.2. Kelas `QuadTreeNode`

```
#ifndef QUADTREENODE_HPP
#define QUADTREENODE_HPP

#include "ErrorMethods.hpp"
#include <tuple>

extern unsigned char* currImgData;
extern unsigned char* tempImgData;
extern int imgWidth, imgHeight, imgChannels;
ErrorMethod* errorMethod = nullptr;

class QuadTreeNode {

    private:
        int step, x, y, width, height, mode;
        double error, avgR, avgG, avgB;

    public:
        QuadTreeNode() {
            this -> x = 0;
            this -> y = 0;
            this -> width = 0;
            this -> height = 0;
            this -> step = 0;
            this -> error = 0;
            this -> avgR = 0;
            this -> avgG = 0;
            this -> avgB = 0;
        }

        QuadTreeNode(int step,
                     int x,
                     int y,
                     int width,
                     int height,
                     int mode) {

            this -> step = step;
            this -> x = x;
            this -> y = y;
            this -> width = width;
            this -> height = height;
            this -> mode = mode;
            this -> error = calculateError(mode);
            this -> avgR = 0;
            this -> avgG = 0;
            this -> avgB = 0;
        }
```

```
QuadTreeNode& operator=(const QuadTreeNode& other) {
    this -> x = other.x;
    this -> y = other.y;
    this -> width = other.width;
    this -> height = other.height;
    this -> step = other.step;
    this -> error = other.error;
    this -> avgR = other.avgR;
    this -> avgG = other.avgG;
    this -> avgB = other.avgB;

    return *this;
}

int getStep() {return step;}
void setStep(int step) {this -> step = step;}

int getX() {return x;}
void setX(int x) {this -> x = x;}

void setY(int y) {this -> y = y;}
int getY() {return y;}

int getWidth() {return width;}
void setWidth(int width) {this -> width = width;}

int getHeight() {return height;}
void setHeight(int height) {this -> height = height;}

tuple<double, double, double> getAvg() {return {avgR, avgG,
↪  avgB};}

void setAvg(double avgR, double avgG, double avgB) {
    this -> avgR = avgR;
    this -> avgG = avgG;
    this -> avgB = avgB;
}

double getError() {return error;}
void setError(double error) {this -> error = error;}

double calculateError(int mode) {
    if (errorMethod == nullptr) {
        switch(mode) {
            case 1:
                errorMethod = new Variance();
                break;
            case 2:
                errorMethod = new MeanAbsoluteDeviation();
                break;
```

```
                    case 3:
                        errorMethod = new MaxPixelDifference();
                        break;
                    case 4:
                        errorMethod = new Entropy();
                        break;
                    case 5:
                        errorMethod = new SSIM();
                        break;
                }
            }

            if (errorMethod) {
                double errorValue = errorMethod ->
                ↪  calculateError(currImgData, x, y, width,
                ↪  height);
                setAvg(errorMethod -> getAvgR(), errorMethod ->
                ↪  getAvgG(), errorMethod -> getAvgB());
                return errorValue;
            }
            else {
                cout << "Error: Error method not initialized." <<
                ↪  endl;
                return 0;
            }
        }

        void fillRectangle(unsigned char* img) {
            if (!img) return;
            for (int i = x; i < x + height; ++i) {
                for (int j = y; j < y + width; ++j) {
                    int idx = (i * imgWidth + j) * imgChannels;
                    img[idx] = static_cast<unsigned char>(avgR);
                    img[idx + 1] = static_cast<unsigned char>(avgG);
                    img[idx + 2] = static_cast<unsigned char>(avgB);
                }
            }
        }

        void fillCurrRectangle() {
            fillRectangle(currImgData);
        }

        void fillTempRectangle() {
            fillRectangle(tempImgData);
        }
};

#endif
```

Kelas `QuadTreeNode` berisi program yang merepresentasikan simpul dalam struktur data quadtree untuk keperluan kompresi citra. Setiap simpul menyimpan informasi mengenai posisi, ukuran, nilai rata-rata warna, serta nilai error berdasarkan metode evaluasi tertentu (error metric). Kelas ini juga menyediakan fungsionalitas untuk menghitung error, mengisi area gambar dengan warna rata-rata, serta mendukung operator penugasan.

Konstruktor utama pada kelas ini adalah `QuadTreeNode(int step, int x, int y, int width, int height, int mode)`, yang akan mengisi atribute dari objek tersebut dengan masing-masing parameter tersebut. Dalam konstruktor tersebut, akan juga langsung dihitung nilai error dari objek tersebut dengan memanggil fungsi lain, yaitu `void calculateError(int mode)`. Dalam proses penghitungan kalkulasi error tersebut, juga dihitung nilai rata-rata dari dari tiap kanal pada pixel-pixel yang terdapat pada blok tersebut. Pada implementasi kelas, terdapat fungsi untuk mewarnai blok dari quadtree dengan warga rata-rata dari blok tersebut, yakni fungsi `void fillRectangle(unsigned char* image)`. Selain itu, terdapat beberapa fungsi-fungsi lain untuk keperluan implementasi, seperti fungsi-fungsi *setter*, *getter*.

3.3. Kelas `ErrorMethod`

```hpp
#ifndef ERROR_METHODS_HPP
#define ERROR_METHODS_HPP

#include "Image.hpp"
#include <unordered_map>

// Global variables for image data
extern int imgWidth, imgHeight, imgChannels;
extern unsigned char* currImgData;

class ErrorMethod {
    protected:
        double upperThreshold;
        double lowerThreshold;
        double avgR, avgG, avgB;

    public:
        virtual double calculateError(unsigned char* currImgData,
        ↪  int x, int y, int width, int height) = 0;
        double getAvgR() const { return avgR; }
        double getAvgG() const { return avgG; }
        double getAvgB() const { return avgB; }
        double getUpperThreshold() const { return upperThreshold; }
        double getLowerThreshold() const { return lowerThreshold; }
};

#endif
```

Kelas `ErrorMethod` digunakan untuk mengkuantifikasi error pada suatu blok, yang mana hasilnya akan dibandingkan dengan variabel `threshold`. Apabila nilai error yang dikembalikan lebih besar daripada threshold, maka blok tersebut akan dibagi kembali menjadi 4 blok. Sebaliknya, apabila nilai error yang dikembalikan lebih kecil, maka blok tersebut tidak perlu dipecah lebih lanjut.

Dalam implementasinya, terdapat 5 kelas turunan dari `ErrorMethod`, yakni kelas `Variance`, `MeanAbsoluteDeviation`, `MaxPixelDifference`, `Entropy`, dan `SSIM`. Masing-masing kelas turunan tersebut digunakan untuk menghitung nilai error pada suatu blok sesuai dengan mode yang dipilih. Berikut implementasi dari metode perhitungan error yang wajib:

3.3.1 Variance

```
// This code still remains in "ErrorMethods.hpp"

class Variance : public ErrorMethod {

    private:
        long long** prefsumR;
        long long** prefsumG;
        long long** prefsumB;
        long long** prefsumR2;
        long long** prefsumG2;
        long long** prefsumB2;

    public:
        Variance() {
            upperThreshold = 127.5 * 127.5;
            lowerThreshold = 0;

            prefsumR = new long long*[imgHeight];
            prefsumG = new long long*[imgHeight];
            prefsumB = new long long*[imgHeight];
            prefsumR2 = new long long*[imgHeight];
            prefsumG2 = new long long*[imgHeight];
            prefsumB2 = new long long*[imgHeight];

            for (int i = 0; i < imgHeight; ++i) {
                prefsumR[i] = new long long[imgWidth];
                prefsumG[i] = new long long[imgWidth];
                prefsumB[i] = new long long[imgWidth];
                prefsumR2[i] = new long long[imgWidth];
                prefsumG2[i] = new long long[imgWidth];
                prefsumB2[i] = new long long[imgWidth];
            }

            for (int i = 0; i < imgHeight; i++) {
                for (int j = 0; j < imgWidth; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;

                    prefsumR[i][j] = currImgData[idx + 0];
                    prefsumG[i][j] = currImgData[idx + 1];
                    prefsumB[i][j] = currImgData[idx + 2];
```

```
                prefsumR2[i][j] = currImgData[idx + 0] *
                ↪  currImgData[idx + 0];
                prefsumG2[i][j] = currImgData[idx + 1] *
                ↪  currImgData[idx + 1];
                prefsumB2[i][j] = currImgData[idx + 2] *
                ↪  currImgData[idx + 2];

                if (i == 0 && j == 0) continue;
                else if (i == 0) {
                    prefsumR[i][j] += prefsumR[i][j - 1];
                    prefsumG[i][j] += prefsumG[i][j - 1];
                    prefsumB[i][j] += prefsumB[i][j - 1];
                    prefsumR2[i][j] += prefsumR2[i][j - 1];
                    prefsumG2[i][j] += prefsumG2[i][j - 1];
                    prefsumB2[i][j] += prefsumB2[i][j - 1];
                }
                else if (j == 0) {
                    prefsumR[i][j] += prefsumR[i - 1][j];
                    prefsumG[i][j] += prefsumG[i - 1][j];
                    prefsumB[i][j] += prefsumB[i - 1][j];
                    prefsumR2[i][j] += prefsumR2[i - 1][j];
                    prefsumG2[i][j] += prefsumG2[i - 1][j];
                    prefsumB2[i][j] += prefsumB2[i - 1][j];
                }
                else {
                    prefsumR[i][j] += prefsumR[i - 1][j] +
                    ↪  prefsumR[i][j - 1] - prefsumR[i - 1][j
                    ↪  - 1];
                    prefsumG[i][j] += prefsumG[i - 1][j] +
                    ↪  prefsumG[i][j - 1] - prefsumG[i - 1][j
                    ↪  - 1];
                    prefsumB[i][j] += prefsumB[i - 1][j] +
                    ↪  prefsumB[i][j - 1] - prefsumB[i - 1][j
                    ↪  - 1];
                    prefsumR2[i][j] += prefsumR2[i - 1][j] +
                    ↪  prefsumR2[i][j - 1] - prefsumR2[i -
                    ↪  1][j - 1];
                    prefsumG2[i][j] += prefsumG2[i - 1][j] +
                    ↪  prefsumG2[i][j - 1] - prefsumG2[i -
                    ↪  1][j - 1];
                    prefsumB2[i][j] += prefsumB2[i - 1][j] +
                    ↪  prefsumB2[i][j - 1] - prefsumB2[i -
                    ↪  1][j - 1];
                }
            }
        }
    }

    ~Variance() {
        for (int i = 0; i < imgHeight; ++i) {
            delete[] prefsumR[i];
            delete[] prefsumG[i];
            delete[] prefsumB[i];
            delete[] prefsumR2[i];
            delete[] prefsumG2[i];
            delete[] prefsumB2[i];
```

```
        }
        delete[] prefsumR;
        delete[] prefsumG;
        delete[] prefsumB;
        delete[] prefsumR2;
        delete[] prefsumG2;
        delete[] prefsumB2;
    }

    double calculateError(unsigned char* currImgData, int row,
    ↪  int col, int width, int height) override {
        long long sumR, sumG, sumB;
        long long sumR2, sumG2, sumB2;

        if (width == 0 || height == 0) {
            return 0;
        }

        if (row == 0 && col == 0) {
            sumR = prefsumR[row + height - 1][col + width - 1];
            sumG = prefsumG[row + height - 1][col + width - 1];
            sumB = prefsumB[row + height - 1][col + width - 1];

            sumR2 = prefsumR2[row + height - 1][col + width -
            ↪  1];
            sumG2 = prefsumG2[row + height - 1][col + width -
            ↪  1];
            sumB2 = prefsumB2[row + height - 1][col + width -
            ↪  1];
        }
        else if (row == 0) {
            sumR = prefsumR[row + height - 1][col + width - 1]
            ↪  - prefsumR[row + height - 1][col - 1];
            sumG = prefsumG[row + height - 1][col + width - 1]
            ↪  - prefsumG[row + height - 1][col - 1];
            sumB = prefsumB[row + height - 1][col + width - 1]
            ↪  - prefsumB[row + height - 1][col - 1];

            sumR2 = prefsumR2[row + height - 1][col + width -
            ↪  1] - prefsumR2[row + height - 1][col - 1];
            sumG2 = prefsumG2[row + height - 1][col + width -
            ↪  1] - prefsumG2[row + height - 1][col - 1];
            sumB2 = prefsumB2[row + height - 1][col + width -
            ↪  1] - prefsumB2[row + height - 1][col - 1];
        }
        else if (col == 0) {
            sumR = prefsumR[row + height - 1][col + width - 1]
            ↪  - prefsumR[row - 1][col + width - 1];
            sumG = prefsumG[row + height - 1][col + width - 1]
            ↪  - prefsumG[row - 1][col + width - 1];
            sumB = prefsumB[row + height - 1][col + width - 1]
            ↪  - prefsumB[row - 1][col + width - 1];

            sumR2 = prefsumR2[row + height - 1][col + width -
            ↪  1] - prefsumR2[row - 1][col + width - 1];
            sumG2 = prefsumG2[row + height - 1][col + width -
            ↪  1] - prefsumG2[row - 1][col + width - 1];
```

```
                sumB2 = prefsumB2[row + height - 1][col + width -
                ↪  1] - prefsumB2[row - 1][col + width - 1];
            }
            else {
                sumR = prefsumR[row + height - 1][col + width - 1]
                ↪  - prefsumR[row - 1][col + width - 1] -
                ↪  prefsumR[row + height - 1][col - 1] +
                ↪  prefsumR[row - 1][col - 1];
                sumG = prefsumG[row + height - 1][col + width - 1]
                ↪  - prefsumG[row - 1][col + width - 1] -
                ↪  prefsumG[row + height - 1][col - 1] +
                ↪  prefsumG[row - 1][col - 1];
                sumB = prefsumB[row + height - 1][col + width - 1]
                ↪  - prefsumB[row - 1][col + width - 1] -
                ↪  prefsumB[row + height - 1][col - 1] +
                ↪  prefsumB[row - 1][col - 1];

                sumR2 = prefsumR2[row + height - 1][col + width -
                ↪  1] - prefsumR2[row - 1][col + width - 1] -
                ↪  prefsumR2[row + height - 1][col - 1] +
                ↪  prefsumR2[row - 1][col - 1];
                sumG2 = prefsumG2[row + height - 1][col + width -
                ↪  1] - prefsumG2[row - 1][col + width - 1] -
                ↪  prefsumG2[row + height - 1][col - 1] +
                ↪  prefsumG2[row - 1][col - 1];
                sumB2 = prefsumB2[row + height - 1][col + width -
                ↪  1] - prefsumB2[row - 1][col + width - 1] -
                ↪  prefsumB2[row + height - 1][col - 1] +
                ↪  prefsumB2[row - 1][col - 1];
            }

            double n = width * height;
            avgR = sumR / n;
            avgG = sumG / n;
            avgB = sumB / n;
            double varianceR = (sumR2 / n) - (avgR * avgR);
            double varianceG = (sumG2 / n) - (avgG * avgG);
            double varianceB = (sumB2 / n) - (avgB * avgB);

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

3.3.2 Mean Absolute Deviation

```
// This code still remains in "ErrorMethods.hpp"

class MeanAbsoluteDeviation : public ErrorMethod {
    public:
        MeanAbsoluteDeviation() {
            upperThreshold = 127.5;
            lowerThreshold = 0;
        }
```

```
        double calculateError(unsigned char* currImgData, int x,
        ↪  int y, int width, int height) override {
            double sumAbsDevR = 0, sumAbsDevG = 0, sumAbsDevB = 0;
            double sumR = 0, sumG = 0, sumB = 0;

            // First pass: calculate average RGB values
            for (int i = x; i < x + height; i++) {
                for (int j = y; j < y + width; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;

                    uint8_t r = currImgData[idx + 0];
                    uint8_t g = currImgData[idx + 1];
                    uint8_t b = currImgData[idx + 2];
                    sumR += r;
                    sumG += g;
                    sumB += b;
                }
            }
            int n = width * height;
            avgR = sumR / n;
            avgG = sumG / n;
            avgB = sumB / n;

            // Second pass: calculate absolute deviations
            for (int i = x; i < x + height; i++) {
                for (int j = y; j < y + width; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;

                    uint8_t r = currImgData[idx + 0];
                    uint8_t g = currImgData[idx + 1];
                    uint8_t b = currImgData[idx + 2];

                    sumAbsDevR += fabs(r - avgR);
                    sumAbsDevG += fabs(g - avgG);
                    sumAbsDevB += fabs(b - avgB);
                }
            }
            double madR = sumAbsDevR / n;
            double madG = sumAbsDevG / n;
            double madB = sumAbsDevB / n;

            return (madR + madG + madB) / 3.0;
        }
};
```

3.3.3 Max Pixel Difference

```
// This code still remains in "ErrorMethods.hpp"

class MaxPixelDifference : public ErrorMethod {
    public:
        MaxPixelDifference() {
            upperThreshold = 255.0;
            lowerThreshold = 0;
        }
```

```
        double calculateError(unsigned char* currImgData, int x,
        ↪  int y, int width, int height) override {
            uint8_t minR = 255, minG = 255, minB = 255;
            uint8_t maxR = 0, maxG = 0, maxB = 0;
            double sumR = 0, sumG = 0, sumB = 0;

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

                    uint8_t r = currImgData[idx + 0];
                    uint8_t g = currImgData[idx + 1];
                    uint8_t b = currImgData[idx + 2];

                    // Find min and max values for each channel
                    minR = std::min(minR, r);
                    minG = std::min(minG, g);
                    minB = std::min(minB, b);

                    maxR = std::max(maxR, r);
                    maxG = std::max(maxG, g);
                    maxB = std::max(maxB, b);

                    sumR += r;
                    sumG += g;
                    sumB += b;
                }
            }

            double diffR = maxR - minR;
            double diffG = maxG - minG;
            double diffB = maxB - minB;

            int n = width * height;
            avgR = sumR / n;
            avgG = sumG / n;
            avgB = sumB / n;

            return (diffR + diffG + diffB) / 3.0;
        }
};
```

3.3.4 Entropy

```
// This code still remains in "ErrorMethods.hpp"

class Entropy : public ErrorMethod {
    public:
        Entropy() {
            // (max entropy for 8-bit values, log2(256) = 8)
            upperThreshold = 8.0;
            lowerThreshold = 0;
        }
```

```
        double calculateError(unsigned char* currImgData, int x,
        ↪  int y, int width, int height) override {
            std::unordered_map<int, int> histR, histG, histB;
            double sumR = 0, sumG = 0, sumB = 0;
            int n = width * height;

            // Step 1: Compute average values
            for (int i = x; i < x + height; i++) {
                for (int j = y; j < y + width; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;

                    uint8_t r = currImgData[idx + 0];
                    uint8_t g = currImgData[idx + 1];
                    uint8_t b = currImgData[idx + 2];
                    sumR += r;
                    sumG += g;
                    sumB += b;
                }
            }
            avgR = sumR / n;
            avgG = sumG / n;
            avgB = sumB / n;

            // Step 2: Build difference histograms
            for (int i = x; i < x + height; i++) {
                for (int j = y; j < y + width; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;

                    int dr = static_cast<int>(currImgData[idx + 0])
                    ↪  - static_cast<int>(avgR);
                    int dg = static_cast<int>(currImgData[idx + 1])
                    ↪  - static_cast<int>(avgG);
                    int db = static_cast<int>(currImgData[idx + 2])
                    ↪  - static_cast<int>(avgB);
                    histR[dr]++;
                    histG[dg]++;
                    histB[db]++;
                }
            }

            // Step 3: Compute entropy
            auto computeEntropy = [&](const std::unordered_map<int,
            ↪  int>& hist) -> double {
                double entropy = 0;
                for (const auto& x : hist) {
                    int count = x.second;
                    double p = static_cast<double>(count) / n;
                    entropy -= p * std::log2(p);
                }
                return entropy;
            };
            double entropyR = computeEntropy(histR);
            double entropyG = computeEntropy(histG);
            double entropyB = computeEntropy(histB);
            return (entropyR + entropyG + entropyB) / 3.0;
        }
};
```

3.4. Penjelasan Implementasi

Logika utama implementasi terdapat pada *method* `void performQuadtree()` pada kelas `QuadTree`, yang merupakan fungsi peran utama dalam menjalankan proses kompresi citra menggunakan algoritma quadtree.

Fungsi ini bekerja dengan menggunakan algoritma Breadth-First Search (BFS), yaitu pendekatan traversal *level-by-level* untuk membentuk simpul-simpul Quadtree. Pendekatan BFS lebih dipilih dibanding pendekatan Rekursi, karena pendekatan BFS lebih memungkinkan untuk pembentukan frame GIF. Dalam konteks ini, algoritma BFS digunakan untuk memastikan bahwa pembentukan simpul dilakukan secara menyeluruh pada tiap tingkat pembagian sebelum ke tingkat upa-divisi berikutnya sehingga memungkinkan pembentukan frame GIF.

Proses dimulai dengan menganalisis seluruh blok citra pada level akar (*root*), yang merepresentasikan keseluruhan area gambar input. Analisis ini bertujuan untuk mengevaluasi apakah blok tersebut telah memenuhi kriteria keseragaman berdasarkan nilai ambang (*threshold*) yang diberikan dan apakah ukurannya telah mencapai batas minimum blok (*minBlock*). Lebih jelasnya, untuk tiap blok :

- Jika blok tersebut tidak memenuhi kriteria tersebut, maka proses pembagian akan dilakukan menggunakan prinsip dasar algoritma *Divide and Conquer*. Blok tersebut akan dibagi menjadi 4 blok dengan ukuran setengah dari ukuran saat ini. Lebih jelasnya, blok dengan posisi pojok kiri atas pada koordinat `(row, col)` dengan ukuran `heigth` × `width` akan dipecah menjadi :

  1. Blok kiri atas:
     Koordinat pojok kiri atas `(row, col)` dengan ukuran `(height / 2)` × `(width / 2)`.
  2. Blok kanan atas:
     Koordinat pojok kiri atas `(row, col + width / 2)` dengan ukuran `(height / 2)` × `(width - width / 2)`.
  3. Blok kiri bawah:
     Koordinat pojok kiri atas `(row + height / 2, col)` dengan ukuran `(height - height / 2)` × `(width / 2)`.
  4. Blok kanan bawah:
     Koordinat pojok kiri atas `(row + height / 2, col + width / 2)` dengan ukuran `(height - height / 2)` × `(width - width / 2)`.

- Jika blok tersebut sudah memenuhi nilai ambang atau ukurannya telah mencapai batas minimum, blok tidak akan dipecah, dan blok tersebut akan diwarnai dengan warna rataan dari seluruh pixel yang ada pada blok tersebut.

Setelah pembagian dilakukan, upa-blok yang dihasilkan dimasukkan ke dalam sebuah struktur `Queue` sebagai bagian dari implementasi algoritma BFS. Dengan pendekatan BFS, pemrosesan simpul dilakukan secara melebar (*level-order traversal*), dengan seluruh simpul pada satu tingkat kedalaman diproses terlebih dahulu sebelum ke tingkat berikutnya. Ini berbeda dengan pendekatan rekursif biasa (seperti Depth-First Search) karena BFS melacak pembentukan Quadtree secara lebih sistematis dan memudahkan integrasi dengan proses pembuatan GIF.

3.5. Penjelasan Bonus yang diimplementasikan

Secara garis besar, seluruh bonus yang ada telah dikerjakan pada tugas kecil ini. Berikut penjelasan terkait masing-masing implementasinya.

3.5.1 Target Compression Percentage

Untuk bonus *Compression Percentage*, digunakan algoritma *binary search* untuk menemukan nilai ambang batas (threshold) yang paling sesuai agar hasil kompresi mendekati target rasio kompresi yang diinginkan. Karena pada kompresi Quadtree terdapat hubungan monoton antara nilai threshold dan ukuran hasil kompresi, yakni semakin besar threshold maka detail gambar yang dipertahankan semakin sedikit dan ukuran file semakin kecil, dan sebaliknya. Kondisi ini sangat cocok untuk diterapkan *binary search*. Dengan menentukan batas bawah dan atas threshold, algoritma dapat menebak nilai tengah secara iteratif, kemudian menjalankan proses kompresi dan mengukur hasilnya untuk dibandingkan dengan target rasio.

```
// This code already mentioned too in "QuadTree.hpp"

void performBinserQuadTree(double ratio) {
    ErrorMethod* errorMethod = nullptr;
    double lowerThreshold = 0.0, upperThreshold = 0.0;

    switch(mode) {
        case 1:
            errorMethod = new Variance();
            break;
        case 2:
            errorMethod = new MeanAbsoluteDeviation();
            break;
        case 3:
            errorMethod = new MaxPixelDifference();
            break;
        case 4:
            errorMethod = new Entropy();
            break;
        case 5:
            errorMethod = new SSIM();
            break;
        default:
            errorMethod = new Variance();
            break;
    }

    if (errorMethod)
    {
        lowerThreshold = errorMethod->getLowerThreshold();
        upperThreshold = errorMethod->getUpperThreshold();
        delete errorMethod;
    }

    size_t initImageSize = Image::getOriginalSize(inputPath);
    size_t targetImageSize = initImageSize - (initImageSize *
    ↪  ratio);
```

```
    double l = lowerThreshold, r = upperThreshold;
    double bestThreshold = -1;
    lastImg = false;

    for (int i = 1; i <= 13; i++) {
        double mid = (l + r) / 2;
        threshold = mid;
        performQuadTree();

        size_t currImgSize = Image::getEncodedSize(currImgData,
        ↪  imgWidth, imgHeight, inputExtension, imgChannels);

        if (currImgSize <= targetImageSize) {
            bestThreshold = mid;
            r = mid;
        }
        else l = mid;

        memcpy(currImgData, initImgData, imgWidth * imgHeight *
        ↪  imgChannels);
    }

    if (bestThreshold == -1) bestThreshold = threshold;

    lastImg = true;
    threshold = bestThreshold;
    performQuadTree();

    size_t currImgSize = Image::getEncodedSize(currImgData,
    ↪  imgWidth, imgHeight, inputExtension, imgChannels);
}
```

Setiap hasil kompresi dari threshold yang diuji akan dibandingkan dengan rasio target, dan selanjutnya arah pencarian disesuaikan: jika hasil kompresi terlalu besar (rasio terlalu kecil), maka threshold dinaikkan; jika terlalu kecil (rasio terlalu besar), maka threshold diturunkan. Proses ini berlanjut hingga ditemukan nilai threshold yang memberikan hasil kompresi paling mendekati target dalam toleransi tertentu. Pendekatan ini sangat efisien, karena hanya memerlukan $O(log\ T)$, dimana T sebagai rentang nilai threshold.

3.5.2 SSIM

Structural Similarity Index (SSIM) merupakan sebuah metode evaluasi kualitas citra yang menilai tingkat kemiripan struktural antara bagian-bagian gambar. Tidak seperti metrik kesalahan konvensional yang hanya memperhatikan perbedaan nilai piksel secara numerik, SSIM mempertimbangkan aspek luminansi, kontras, dan struktur untuk meniru cara kerja sistem penglihatan manusia dalam menilai kualitas visual.

$$SSIM_c\ (x, y) = \frac{(2\ \mu_{x,c}\ \mu_{y,c} + C_1)\ (2\ \sigma_{xy,c} + C_2)}{\left(\mu_{x,c}^2 + \mu_{y,c}^2 + C_1\right)\left(\sigma_{x,c}^2 + \sigma_{y,c}^2 + C_2\right)}$$

Dalam konteks tugas ini, SSIM digunakan untuk menentukan apakah suatu blok gambar cukup seragam sehingga dapat dikompresi tanpa kehilangan detail penting secara visual. Implementasi SSIM dalam program dilakukan secara efisien menggunakan teknik integral image, yang memungkinkan perhitungan rata-rata dan variansi piksel dalam blok dilakukan dengan cepat dan akurat.

Dalam program ini, SSIM diimplementasikan secara lebih efisien dengan penyederhanaan persamaan dan memanfaatkan teknik *Prefix Sum 2D*. Struktur integral image menyimpan akumulasi nilai piksel dan kuadratnya dalam matriks 2D untuk masing-masing kanal warna (R, G, dan B). Hal ini memungkinkan pengambilan nilai total, rata-rata, dan variansi sebuah blok dilakukan dalam waktu konstan, tanpa perlu iterasi langsung ke tiap piksel di blok tersebut. Implementasi ini mengurangi kompleksitas waktu per blok dari $O(w \times h)$ menjadi $O(1)$ sehingga sangat efektif ketika harus menghitung nilai error untuk banyak blok secara berulang dalam proses traversal Quadtree. Pendekatan ini menghasilkan peningkatan performa yang signifikan tanpa mengorbankan akurasi perhitungan SSIM.

```
class SSIM : public ErrorMethod {
    private:
        const double C2 = 58.5225; // (0.03 * 255) power 2
        double** sumR;
        double** sumG;
        double** sumB;
        double** sumR2;
        double** sumG2;
        double** sumB2;

    public:
        SSIM() {
            upperThreshold = 1.0;
            lowerThreshold = 0;

            sumR = new double*[imgHeight];
            sumG = new double*[imgHeight];
            sumB = new double*[imgHeight];
            sumR2 = new double*[imgHeight];
            sumG2 = new double*[imgHeight];
            sumB2 = new double*[imgHeight];

            for (int i = 0; i < imgHeight; ++i) {

                sumR[i] = new double[imgWidth];
                sumG[i] = new double[imgWidth];
                sumB[i] = new double[imgWidth];
                sumR2[i] = new double[imgWidth];
                sumG2[i] = new double[imgWidth];
                sumB2[i] = new double[imgWidth];
            }

            for (int i = 0; i < imgHeight; i++) {
                for (int j = 0; j < imgWidth; j++) {
                    int idx = (i * imgWidth + j) * imgChannels;
```

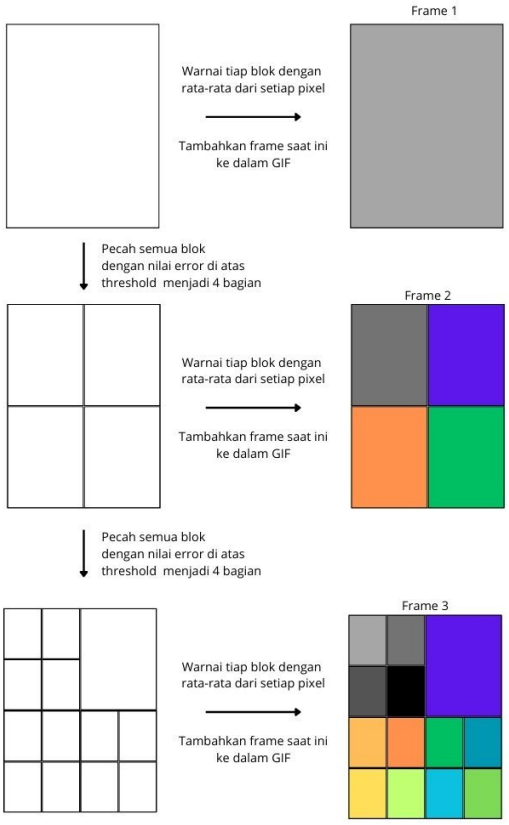
```
                sumR[i][j] = currImgData[idx + 0];
                sumG[i][j] = currImgData[idx + 1];
                sumB[i][j] = currImgData[idx + 2];
                sumR2[i][j] = currImgData[idx + 0] *
                ↪  currImgData[idx + 0];
                sumG2[i][j] = currImgData[idx + 1] *
                ↪  currImgData[idx + 1];
                sumB2[i][j] = currImgData[idx + 2] *
                ↪  currImgData[idx + 2];

                if (i == 0 && j == 0) {
                    continue;
                }
                else if (i == 0) {
                    sumR[i][j] += sumR[i][j - 1];
                    sumG[i][j] += sumG[i][j - 1];
                    sumB[i][j] += sumB[i][j - 1];
                    sumR2[i][j] += sumR2[i][j - 1];
                    sumG2[i][j] += sumG2[i][j - 1];
                    sumB2[i][j] += sumB2[i][j - 1];
                }
                else if (j == 0) {
                    sumR[i][j] += sumR[i - 1][j];
                    sumG[i][j] += sumG[i - 1][j];
                    sumB[i][j] += sumB[i - 1][j];
                    sumR2[i][j] += sumR2[i - 1][j];
                    sumG2[i][j] += sumG2[i - 1][j];
                    sumB2[i][j] += sumB2[i - 1][j];
                }
                else {
                    sumR[i][j] += sumR[i - 1][j] + sumR[i][j -
                    ↪  1] - sumR[i - 1][j - 1];
                    sumG[i][j] += sumG[i - 1][j] + sumG[i][j -
                    ↪  1] - sumG[i - 1][j - 1];
                    sumB[i][j] += sumB[i - 1][j] + sumB[i][j -
                    ↪  1] - sumB[i - 1][j - 1];
                    sumR2[i][j] += sumR2[i - 1][j] + sumR2[i][j
                    ↪  - 1] - sumR2[i - 1][j - 1];
                    sumG2[i][j] += sumG2[i - 1][j] + sumG2[i][j
                    ↪  - 1] - sumG2[i - 1][j - 1];
                    sumB2[i][j] += sumB2[i - 1][j] + sumB2[i][j
                    ↪  - 1] - sumB2[i - 1][j - 1];
                }
            }
        }
    }

    double calculateError(unsigned char* currImgData, int x,
    ↪  int y, int width, int height) override {
        int x1 = x;
        int y1 = y;
        int x2 = x + height - 1;
        int y2 = y + width - 1;
        int n = width * height;

        if (n == 0) return 0;
```

```
            if (x1 < 0 || y1 < 0 || x2 >= imgHeight || y2 >=
            ↪  imgWidth) {
                return 0;
            }
            if (x1 > x2 || y1 > y2) return 0;

            // Helper lambda to extract sum from integral image
            auto getSum = [&](double** integral) -> double {
                double total = integral[x2][y2];
                if (x1 > 0) total -= integral[x1 - 1][y2];
                if (y1 > 0) total -= integral[x2][y1 - 1];
                if (x1 > 0 && y1 > 0) total += integral[x1 - 1][y1
                ↪  - 1];
                return total;
            };

            double sumRVal = getSum(sumR);
            double sumGVal = getSum(sumG);
            double sumBVal = getSum(sumB);
            double sumR2Val = getSum(sumR2);
            double sumG2Val = getSum(sumG2);
            double sumB2Val = getSum(sumB2);

            double meanR = sumRVal / n;
            double meanG = sumGVal / n;
            double meanB = sumBVal / n;

            double varR = (sumR2Val / n) - (meanR * meanR);
            double varG = (sumG2Val / n) - (meanG * meanG);
            double varB = (sumB2Val / n) - (meanB * meanB);

            // In the standard SSIM formula, when comparing
            // an image x with an image y:
            // SSIM(x,y) = [2 x y + C1] / [ x² + y² + C1] · [2 xy +
            ↪  C2] / [ x² + y² + C2]

            // When y is a uniform block with the same mean as x //
            ↪  ( y = x and y = 0), resulting y = 0 and
            // this simplifies to:
            // SSIM(x,uniform) = [2 x² + C1] / [2 x² + C1] · [C2] /
            ↪  [ x² + C2]

            // The first term equals 1, so:
            // SSIM(x,uniform) = C2 / ( x² + C2)

            double ssimR = C2 / (varR + C2);
            double ssimG = C2 / (varG + C2);
            double ssimB = C2 / (varB + C2);

            avgR = meanR;
            avgG = meanG;
            avgB = meanB;

            return (1.0 - ssimR + 1.0 - ssimG + 1.0 - ssimB) / 3.0;
        }
};
```

3.5.3 GIF

Untuk bonus GIF, perlu adanya penyesuaian pendekatan quadtree. Quadtree umumnya diimplementasikan menggunakan pendekatan rekursi (DFS), di mana subdivisi blok citra dilakukan hingga ke simpul terdalam terlebih dahulu sebelum kembali ke atas. Namun, pendekatan ini kurang ideal untuk menghasilkan visualisasi bertahap, karena simpul-simpul pada level yang sama tidak diproses secara bersamaan sehingga animasi yang dihasilkan menjadi kurang informatif dan tidak terstruktur secara progresif.

Sesuai dengan penjelasan implementasi sebelumnya, digunakan pendekatan breadth-first search (BFS) untuk mengatasi hal tersebut. Dengan BFS, seluruh simpul pada satu tingkat kedalaman akan diproses terlebih dahulu sebelum ke tingkat berikutnya. Hal ini memungkinkan proses subdivisi dilakukan secara bertahap dan berurutan dari atas ke bawah, yang sangat cocok untuk visualisasi animasi. Setiap kali satu level selesai diproses, citra hasil sementara dapat disimpan sebagai satu frame dalam file GIF.



Gambar 4: Ilustrasi pembentukan GIF

Dengan cara ini, pengguna dapat melihat secara jelas bagaimana struktur Quadtree terbentuk dari blok besar ke blok-blok kecil secara berurutan dan visualisasi menjadi lebih intuitif serta mudah dipahami.

Bab IV
Eksperimen

4.1. Pengujian Default (Valid & Windows Based)

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: jpg_medium.jpg (736 x 736) 110.77 KB. Path:
↪  D:\test\jpg_medium.jpg
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 150.00
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\jpg_medium_out.jpg
[7/7] GIF Path: D:\test\jpg_medium_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 1803 ms
[-] Initial size: 113427 bytes (110.77 KB)
[-] Final size: 112970 bytes (110.32 KB)
[-] Compression percentage: 0.40 %
[-] Quadtree depth: 10
[-] Quadtree node: 93961
```

Gambar 5: Kasus Uji `default-1` : Input `JPG` berukuran sedang, Mode 1

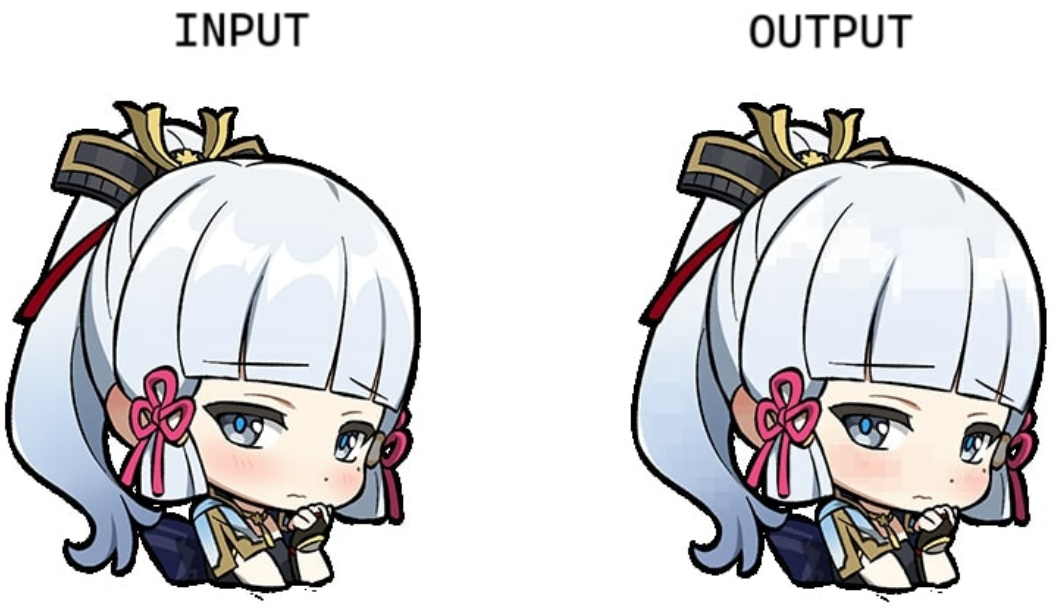
```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: jpg_big.jpg (2560 x 1440) 923.84 KB. Path: D:\test\jpg_big.jpg
[2/7] Error Measurement Method: Mean Absolute Deviation (MAD)
[3/7] Threshold: 10.00
[4/7] Minimum Block Size: 4
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\jpg_big_out.jpg
[7/7] GIF Path: D:\test\jpg_big_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 11417 ms
[-] Initial size: 946008 bytes (923.84 KB)
[-] Final size: 273192 bytes (266.79 KB)
[-] Compression percentage: 71.12 %
[-] Quadtree depth: 11
[-] Quadtree node: 91881
```

INPUT

OUTPUT

Gambar 6: Kasus Uji `default-2` : Input `JPG` berukuran besar, Mode 2

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_small.png (340 x 340) 111.46 KB. Path:
↪  D:\test\png_small.png
[2/7] Error Measurement Method: Max Pixel Difference (MPD)
[3/7] Threshold: 20.00
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\png_small_out.png
[7/7] GIF Path: D:\test\png_small_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 319 ms
[-] Initial size: 114132 bytes (111.46 KB)
[-] Final size: 105238 bytes (102.77 KB)
[-] Compression percentage: 7.79 %
[-] Quadtree depth: 9
[-] Quadtree node: 45697
```



Gambar 7: Kasus Uji `default-1` : Input `PNG` berukuran kecil, Mode 3

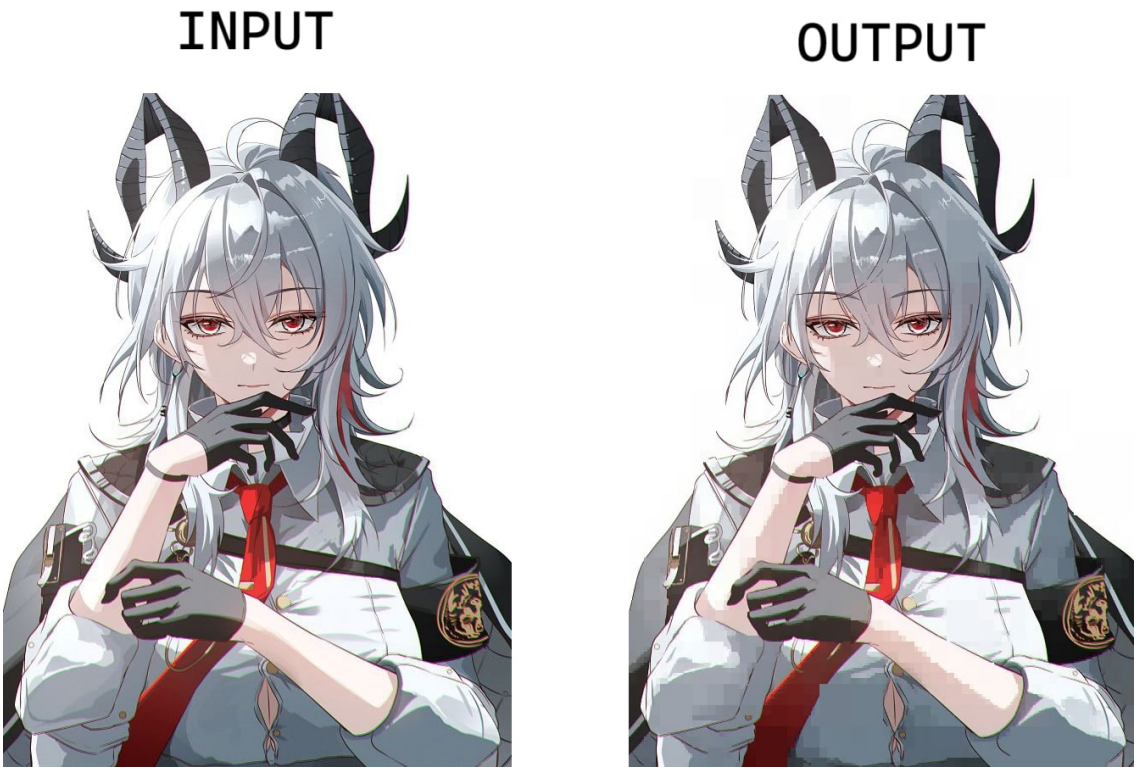
```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_big.png (2174 x 1300) 3683.16 KB. Path:
↪  D:\test\png_big.png
[2/7] Error Measurement Method: Entropy
[3/7] Threshold: 4.00
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\png_big_out.png
[7/7] GIF Path: D:\test\png_big_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 11558 ms
[-] Initial size: 3771556 bytes (3683.16 KB)
[-] Final size: 234803 bytes (229.30 KB)
[-] Compression percentage: 93.77 %
[-] Quadtree depth: 9
[-] Quadtree node: 32897
```

INPUT

OUTPUT

Gambar 8: Kasus Uji `default-1` : Input `PNG` berukuran besar, Mode 4

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_medium.png (600 x 795) 566.48 KB. Path:
↪  D:\test\png_medium.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.75
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\png_medium_out.png
[7/7] GIF Path: D:\test\png_medium_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 1494 ms
[-] Initial size: 580080 bytes (566.48 KB)
[-] Final size: 323002 bytes (315.43 KB)
[-] Compression percentage: 44.32 %
[-] Quadtree depth: 10
[-] Quadtree node: 128077
```

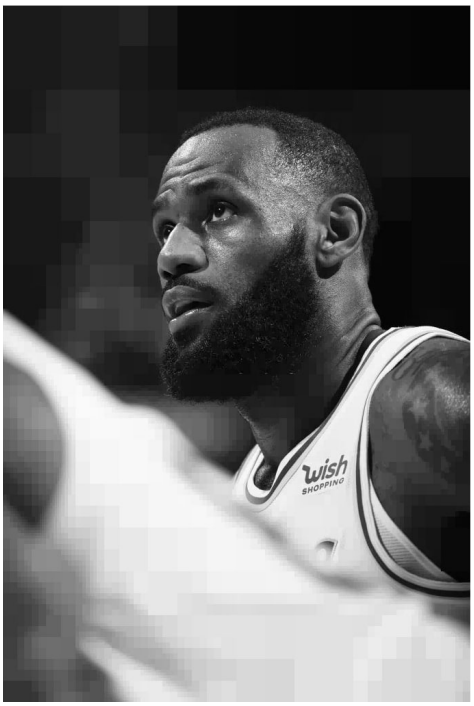


Gambar 9: Kasus Uji `default-5` : Input `PNG` berukuran sedang, Mode 5

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: grayscale.jpg (683 x 1024) 66.87 KB. Path:
↪  D:\test\grayscale.jpg
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.30
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: D:\test\grayscale_out.jpg
[7/7] GIF Path: D:\test\grayscale_out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 1797 ms
[-] Initial size: 68476 bytes (66.87 KB)
[-] Final size: 84079 bytes (82.11 KB)
[-] Compression percentage: -22.79 %
[-] Quadtree depth: 10
[-] Quadtree node: 175253
```

INPUT

OUTPUT



Gambar 10: Kasus Uji `grayscale`

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_medium.png (600 x 795) 566.48 KB. Path:
↪  D:\test\png_medium.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.69
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.25 (25.00%)
[6/7] Output Path: D:\test\png_medium_25.png
[7/7] GIF Path: D:\test\png_medium_25.gif

Input done.
Performing quadtree compression...
Using error method 5 with threshold range: 0.00 to 1.00
Init Size: 580080bytes (566.48 KB)
Target Size: 435060bytes (424.86 KB)
Quadtree compression done.

~ Results ~
[-] Executing time: 3611 ms
[-] Initial size: 580080 bytes (566.48 KB)
[-] Final size: 434960 bytes (424.77 KB)
[-] Compression percentage: 25.02 %
[-] Quadtree depth: 10
[-] Quadtree node: 199541
```

INPUT

OUTPUT

Gambar 11: Kasus Uji `Target Kompresi 0.25`

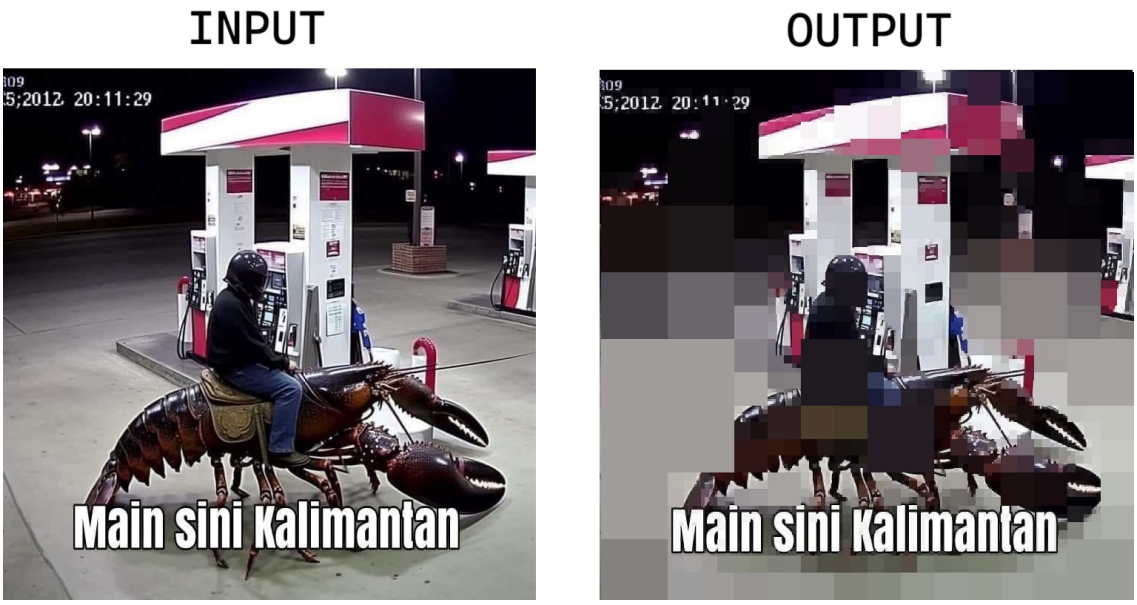
```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_medium.png (600 x 795) 566.48 KB. Path:
↪  D:\test\png_medium.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.69
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.50 (50.00%)
[6/7] Output Path: D:\test\png_medium_50.png
[7/7] GIF Path: D:\test\png_medium_50.gif

Input done.
Performing quadtree compression...
Using error method 5 with threshold range: 0.00 to 1.00
Init Size: 580080bytes (566.48 KB)
Target Size: 290040bytes (283.24 KB)
Quadtree compression done.

~ Results ~
[-] Executing time: 3189 ms
[-] Initial size: 580080 bytes (566.48 KB)
[-] Final size: 290024 bytes (283.23 KB)
[-] Compression percentage: 50.00 %
[-] Quadtree depth: 10
[-] Quadtree node: 110889
```

INPUT

OUTPUT

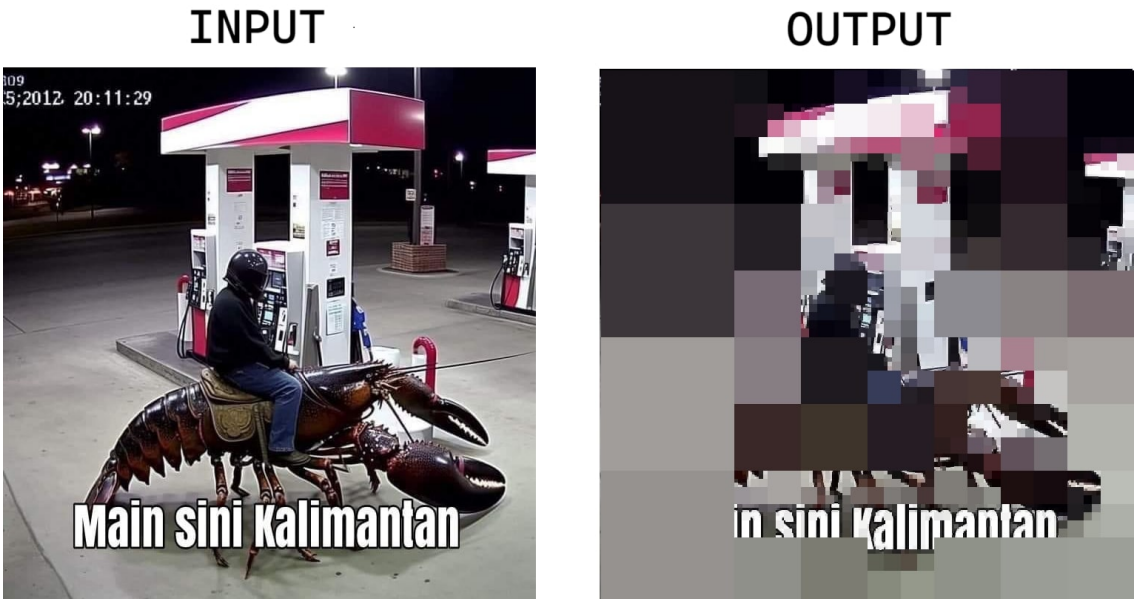
Gambar 12: Kasus Uji `Target Kompresi 0.50`

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_medium.png (600 x 795) 566.48 KB. Path:
↪  D:\test\png_medium.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.69
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.75 (75.00%)
[6/7] Output Path: D:\test\png_medium_75.png
[7/7] GIF Path: D:\test\png_medium_75.gif

Input done.
Performing quadtree compression...
Using error method 5 with threshold range: 0.00 to 1.00
Init Size: 580080bytes (566.48 KB)
Target Size: 145020bytes (141.62 KB)
Quadtree compression done.

~ Results ~
[-] Executing time: 3031 ms
[-] Initial size: 580080 bytes (566.48 KB)
[-] Final size: 144986 bytes (141.59 KB)
[-] Compression percentage: 75.01 %
[-] Quadtree depth: 10
[-] Quadtree node: 45381
```



Gambar 13: Kasus Uji `Target Kompresi 0.75`

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: test.png (720 x 720) 413.91 KB. Path: D:\test\test.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.69
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.90 (90.00%)
[6/7] Output Path: D:\test\test_90.png
[7/7] GIF Path: D:\test\test_90.gif

Input done.
Performing quadtree compression...
Using error method 5 with threshold range: 0.00 to 1.00
Init Size: 423846bytes (413.91 KB)
Target Size: 42384bytes (41.39 KB)
Quadtree compression done.

~ Results ~
[-] Executing time: 2343 ms
[-] Initial size: 423846 bytes (413.91 KB)
[-] Final size: 41612 bytes (40.64 KB)
[-] Compression percentage: 90.18 %
[-] Quadtree depth: 10
[-] Quadtree node: 10681
```



Gambar 14: Kasus Uji `Target Kompresi 0.90`

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: test.png (720 x 720) 413.91 KB. Path: D:\test\test.png
[2/7] Error Measurement Method: Structural Similarity Index (SSIM)
[3/7] Threshold: 0.69
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 1.00 (100.00%)
[6/7] Output Path: D:\test\test_100.png
[7/7] GIF Path: D:\test\test_100.gif

Input done.
Performing quadtree compression...
Using error method 5 with threshold range: 0.00 to 1.00
Init Size: 423846bytes (413.91 KB)
Target Size: 0bytes (0.00 KB)
Quadtree compression done.

~ Results ~
[-] Executing time: 1558 ms
[-] Initial size: 423846 bytes (413.91 KB)
[-] Final size: 16302 bytes (15.92 KB)
[-] Compression percentage: 96.15 %
[-] Quadtree depth: 6
[-] Quadtree node: 1
```

Gambar 15: Kasus Uji `Target Kompresi 1.0`

4.2. Pengujian UNIX (Linux/iOS based)

4.2.1 User Profile

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.jpeg (1024 x 1024) 340.90 KB. Path:
↪  /home/rzi/tucil2/in.jpeg
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 1000.00
[4/7] Minimum Block Size: 8
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: /home/rzi/tucil2/out.jpeg
[7/7] GIF Path: /home/rzi/tucil2/out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 2082 ms
[-] Initial size: 349081 bytes (340.90 KB)
[-] Final size: 29852 bytes (29.15 KB)
[-] Compression percentage: 91.45 %
[-] Quadtree depth: 8
[-] Quadtree node: 2917
```

4.2.2 Mounting to Windows Disk

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.jpeg (1024 x 1024) 340.90 KB. Path: /mnt/d/in.jpeg
[2/7] Error Measurement Method: Entropy
[3/7] Threshold: 4.00
[4/7] Minimum Block Size: 8
[5/7] Target Percentage: 0.00 (disabled)
[6/7] Output Path: /mnt/d/out.jpeg
[7/7] GIF Path: /mnt/d/out.gif

Input done.
Performing quadtree compression...
Quadtree compression done.

~ Results ~
[-] Executing time: 8348 ms
[-] Initial size: 349081 bytes (340.90 KB)
[-] Final size: 141475 bytes (138.16 KB)
[-] Compression percentage: 59.47 %
[-] Quadtree depth: 8
[-] Quadtree node: 59153
```

Catatan: *Path* pada UNIX hanya bisa sampai *user profile* `/home/<user>`, program tidak dapat melakukan *read/write* pada direktori di atasnya (seperti *root directory*).

4.3. Pengujian Kasus Invalid

Blank Path

```
Input :
>> (string kosong)

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows

[!] Error: Classic empty input, coba lagi ya bang :D

>>
```

Path Tidak Sesuai Format

```
Input :
>> Super idol de xiao rong dou mei ni de tian Ba yue
   zheng wu de yang guang dou mei ni yao yan Re ai yi bai
   ling wu du de ni Di di qing chun de zheng liu shui

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows

[!] Error: Format path-nya salah bang.

>>
```

Direktori Tidak Ada

```
Input :
>> D:\test\non-existent-directory\image.jpg

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows

[!] Error: Direktorinya engga ada, coba tambahin dulu deh

>>
```

Gambar Tidak Ditemukan

```
Input :
>> D:\test\non-existent-image.jpg

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows

[!] Error: Ga ada file image-nya bang, lupa taroh ya?

>>
```

Ekstensi Gambar Kosong

```
Input :
>> D:\test\file

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows

[!] Error: Error: Jangan lupa extensionnya yaa... udah gede loh

>>
```

Ekstensi Gambar Tidak Sesuai

```
Input :
>> D:\test\example_webp.webp

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows
[-] /home/<user>/example_in.jpg for Linux
[-] /home/<user>/example_in.jpg or /mnt/<drive>/example_in.jpg for
↪  WSL

[!] Error: Extensionnya cuma .jpg, .jpeg, .png aja yaa...

>>
```

Gambar Corrupt

```
Input :
>> D:\test\corrupted_file.jpg

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Enter input path, supported formats: jpg, jpeg, png.

[?] Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows
[-] /home/<user>/example_in.jpg for Linux
[-] /home/<user>/example_in.jpg or /mnt/<drive>/example_in.jpg for
↪  WSL

[!] Error: Image-nya harus ada minimal 3 channel (RGB), kok ini
↪  cuma 0 channel doang.

>>
```

Metode Penghitungan Error Tidak Sesuai - String

```
Input :
>> Variance

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Enter error measurement method...

[?] Available modes:
[-] 1 ~ Variance
[-] 2 ~ Mean Absolute Deviation (MAD)
[-] 3 ~ Max Pixel Difference (MPD)
[-] 4 ~ Entropy
[-] 5 ~ Structural Similarity Index (SSIM)

[!] Error: Angka doang bolehnya yaa, input-nya jangan aneh-aneh
↪  pls.

>>
```

Metode Penghitungan Error Tidak Sesuai - Angka di Luar Range

```
Input :
>> 10

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Enter error measurement method...

[?] Available modes:
[-] 1 ~ Variance
[-] 2 ~ Mean Absolute Deviation (MAD)
[-] 3 ~ Max Pixel Difference (MPD)
[-] 4 ~ Entropy
[-] 5 ~ Structural Similarity Index (SSIM)

[!] Error: Bolehnya angka 1 and 5 doang, yok literasinya

>>
```

Threshold di Luar Range

```
Input :
>> -69420

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Error Measurement Method: Variance
[3/7] Enter threshold...

[?] Threshold range: 0.00 - 16256.25
[-] Example: 8128.12

[!] Error: Invalid threshold, coba dibaca lagi yaa batasnya dari
↪  0.00 sampai 16256.25 (Inklusif).

>>
```

Threshold Bukan Angka

```
Input :
>> Lorem ipsum dolor sit amet, consectetur adipiscing elit. In
   massa tellus, suscipit eget semper vitae, auctor id odio. Donec
   non euismod urna. Aenean iaculis vitae ipsum sit amet imperdiet.

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Error Measurement Method: Variance
[3/7] Enter threshold...

[?] Threshold range: 0.00 - 16256.25
[-] Example: 8128.12

[!] Error: Angka doang bolehnya yaa, input-nya jangan aneh-aneh
↪  pls.

>>
```

Threshold Multiple Input pada satu baris

```
Input :
>> 0.1 0.1

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Error Measurement Method: Variance
[3/7] Enter threshold...

[?] Threshold range: 0.00 - 16256.25
[-] Example: 8128.12

[!] Error: Bener sih angka doang, tapi gabole multiple input yaa...

>>
```

Minimum Block Negatif

```
Input :
>> -69

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: ininin.png (600 x 795) 566.48 KB. Path:
↪  D:\test\ininin.png
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 69.00
[4/7] Enter minimum block size...

[?] Must not exceed the image area (477000 px)
[-] Example: 8

[!] Error: Sejak kapan area ada yang negatif :) coba lagi ya

>>
```

Minimum Block Terlalu Besar

```
Input :
>> 17701369

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: inin.png (600 x 795) 566.48 KB. Path: D:\test\inin.png
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 69.00
[4/7] Enter minimum block size...

[?] Must not exceed the image area (477000 px)
[-] Example: 8

[!] Error: Block size terlalu besar, maksimal 477000 px yaa.

>>
```

Minimum Block Overflow

```
Input :
69420694206942069420694206942069420694206942069420694206942069420

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: png_medium.png (600 x 795) 566.48 KB. Path:
↪  D:\test\png_medium.png
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 69.00
[4/7] Enter minimum block size...

[?] Must not exceed the image area (477000 px)
[-] Example: 8

[!] Error: Angkanya terlalu besar, coba yang lebih kecil ya

>>
```

Target Kompresi Di Luar Range

```
Input :
1.2

Output :
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Error Method Measurement: Variance
[3/7] Threshold: 69.00
[4/7] Minimum Block Size: 1
[5/7] Enter target percentage...

[?] Decimal (0.0 - 1.0 where 1.0 = 100%)
[-] Example: 0.85 for 85%

[!] Error: Bolehnya di rentang 0.0 sampai 1.0 aja yaa.

>>
```

Catatan:

Validasi kasus *empty* selalu berlaku, validasi kasus *non-numerical*, *multiple*, *out of range* berlaku pada seluruh masukan yang seharusnya bernilai numerikal.

4.4. Pengujian File Output yang Telah Ada

```
Quadpressor ~ made with love by FaRzi :D

~ Tasks ~
[1/7] Image: in.png (600 x 795) 566.48 KB. Path: D:\test\in.png
[2/7] Error Measurement Method: Variance
[3/7] Threshold: 69.00
[4/7] Minimum Block Size: 1
[5/7] Target Percentage: 0.40 (40.00%)
[6/7] Enter image output path...

[?] Output extension must be the same as input. Example:
[-] D:\bla-bla-bla\example_in.jpg for Windows
[-] /home/<user>/example_in.jpg for Linux
[-] /home/<user>/example_in.jpg or /mnt/<drive>/example_in.jpg for WSL

>>  D:\test\out.png

[!] Warning: out.png file udah ada, jadinya bakal di-overwrite.
[?] Yakin ga nih? [Y/N]

>>
```

4.5. Analisis dan Pembahasan

Dalam beberapa kasus pengujian, kami menemukan bahwa persentase kompresi quadtree bernilai negatif (yakni, ukuran file hasil kompresi lebih besar dibandingkan file asli), atau memiliki persentase kompresi yang sangat kecil. Setelah ditelusuri lebih lanjut, hal ini umumnya terjadi pada file masukan berformat `.jpg`. Ketika kami melakukan pembacaan gambar (read) lalu langsung melakukan penulisan ulang (write) tanpa memodifikasi isi pixel sama sekali, kami mendapati bahwa ukuran file hasil penulisan ulang tersebut justru meningkat drastis dibandingkan file aslinya.

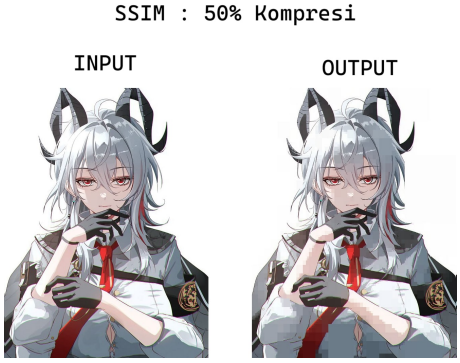
Hal ini terjadi karena format `.jpg` bersifat *lossy* dan menyimpan gambar dengan tingkat kompresi tertentu. File `.jpg` asli yang digunakan kemungkinan besar telah disimpan dengan tingkat kompresi rendah (quality rendah) oleh sumbernya, sedangkan library untuk pembacaan dan penulisan yang kami gunakan, yakni `STB Image` secara default melakukan penulisan `.jpg` dengan kualitas tinggi. Kualitas yang lebih tinggi berarti ukuran file akan lebih besar karena kompresi yang lebih sedikit. Dengan kata lain, meskipun secara visual gambar tidak berubah, namun secara internal struktur data JPEG berbeda, dan ukuran file pun menjadi jauh lebih besar.

Gambar 16: Kasus Uji `default-1` : Input `JPG` berukuran sedang, Mode 1

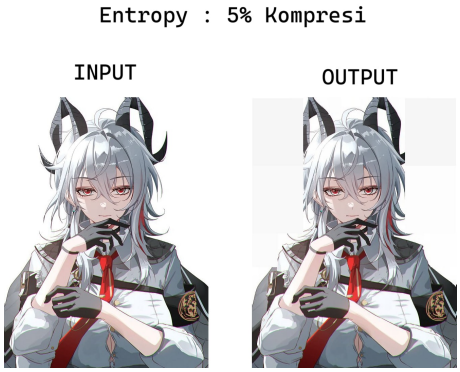
Pada gambar yang ditampilkan, gambar original (kiri), memiliki tekstur atau tingkat *noise* yang tinggi jika dibandingkan dengan gambar hasil kompresi quadtree (kanan). Untuk masukan gambar dengan format `.jpg`, karena adanya perbedaan tingkat kompresi yang sudah ada pada gambar masukan, hasil persentase kompresi quadtree mungkin tidak akan terlalu efektif. Akan teapi, pada gambar dengan format `.png`, metode kompresi yang digunakan adalah metode *Lossless*, yakni metode kompresi tanpa ada data yang hilang. Pada percobaan-percobaan yang melibatkan masukan gambar `.png`, persentase kompresi quadtree berjalan dengan sangat efektif.

Pada input dengan format `.jpg`, untuk menjaga kompresi quadtree agar tetap menghasilan gambar dengan hasil yang serupa dengan gambar asli, dibutuhkan adanya informasi mengenai berapa kualitas kompresi yang digunakan oleh gambar tersebut. Sayangnya, informasi tersebut tidak bisa didapatkan secara langsung dengan cara-cara yang tersedia. Sehingga kami memutuskan untuk melakukan binary search untuk menerka angka kualitas kompresi yang digunakan oleh gambar tersebut. Dengan demikian, gambar input maupun gambar output sama-sama menggunakan nilai kualitas kompresi `.jpg` yang sama, sehingga meningkatkan efektifitas dari kompresi quadtree.

Kami juga mendapati bahwa metode penghitungan error *Entrophy* menghasilkan gambar yang tidak terlalu baik. Dalam beberapa kasus percobaan dengan menggunakan metode entrophy, terjadi kompresi yang cukup signifikan, pada bagian-bagian yang memerlukan detail tinggi. Seperti contohnya pada percobaan bverikut :



Gambar 17: Kasus Uji SSIM - Target Kompresi 50%



Gambar 18: Kasus Uji Entropy - Target Kompresi 5%

Dalam 2 percobaan di atas, dapat terlihat bahwa penghitungan error dengan metode entrophy gagal untuk menjaga detail-detail penting dari suatu blok. Di sisi lain, metode lain, yakni SSIM, berhasil menjaga detail-detail gambar dengan sangat baik, meskipun persentase kompresinya sudah menyentuh di angka 50%.

Bab V
Penutup

5.1. Kesimpulan

Dalam proyek Tugas Kecil 2 IF2211 Strategi Algoritma ini, algoritma *Divide and Conquer* dengan metode Quadtree dan parameter-parameter yang diinginkan dapat digunakan dalam proses kompresi gambar dengan memberikan hasil yang baik secara cepat dan efisien.

Kami berhasil untuk mengimplementasikan *Divide and Conquer* dengan struktur data QuadTree dalam proses kompresi gambar program kami. Selain menghasilkan suatu gambar yang terkompresi, program kami juga dapat menghasilkan `gif` yang dapat memberi *insight* yang lebih mendalam terhadap bagaimana QuadTree bekerja dalam program kompresi gambar ini.

Selain itu, pengguna dapat menentukan target kompresi yang diinginkan agar mendapat kualitas yang maksimal dengan ukuran yang diinginkan, membuat fleksibilitas program dalam menerima perintah dari pengguna.

5.2. Saran

Pelaksanaan Tugas Kecil 2 IF2211 Strategi Algoritma di Semester II (Genap) Tahun 2024/2025 merupakan pengalaman yang sangat berharga bagi kami. Dari pengalaman ini, kami ingin berbagi beberapa saran kepada pembaca yang mungkin akan menghadapi tugas serupa di masa depan:

rzi

Tantangan terbesar sebenarnya dalam tucil dan tubes di IF hanyalah prokas. Sungguh, dunia begitu damai jika kelar tucil dan tubes jauh hari sebelum deadline. Terima kasih banyak farr, semoga yang disemogakan secepatnya tersemogakan.

ryzz

kompresi jpg malesin anjay

Semoga saran-saran ini membantu pembaca dalam menyiapkan diri untuk menangani tugas serupa di masa depan.

5.3. Tautan

Repository program dan GIF yang dihasilkan dapat diakses melalui tautan berikut:

`https://github.com/zirachw/Tucil2_13523004_13523069`

5.4. Lampiran

Tabel 1: Tabel Spesifikasi Tugas Kecil 2

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	✓	

Referensi

Rinaldi Munir. (2025). "Algoritma Divide and Conquer (Bagian 1) (Versi baru 2025)." Diakses pada 27 Maret 2025. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf

Geeks For Geeks. (2025). "Introduction to Divide and Conquer Algorithm." Diakses pada 27 Maret 2025. https://www.geeksforgeeks.org/introduction-to-divide-and-conquer-algorithm/

Yeshwanth N. (2023). "Quadtree: A Spatial Data Structure for Efficient Queries" Diakses pada 2 April 2025. https://medium.com/@yeshsurya/quadtree-a-spatial-data-structure-for-efficient-queries-f4f92958881d

WsCube Tech. (2025). "DFS vs BFS Algorithm (All Differences With Example)" Diakses pada 2 April 2025. https://www.wscubetech.com/resources/dsa/dfs-vs-bfs

Charlie Tangora. (2024). "gif.h Library" Diakses pada 28 Maret 2025. https://github.com/charlietangora/gif-h

Sean Barrett. (2024). "stb_image Library" Diakses pada 28 Maret 2025. github.com/nothings/stb/blob/master/stb_image.h