IF2211 - Strategi Algoritma

Laporan Tugas Kecil 2 Stima: Kompresi Gambar Dengan Metode Quadtree

Samuel Gerrard Hamonangan Girsang	13523064
Nadhif Al Rozin	13523076

Program Studi Teknik Informatika

Sekolah Teknik Elektro dan Informatika

Institut Teknologi Bandung

2025

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.

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

Algoritma yang digunakan pada kompresi gambar metode Quadtree adalah algoritma **Divide and Conquer**. Sebuah gambar direpresentasikan sebagai suatu blok (dalam kasus ini kami representasikan sebagai matriks RGB), kemudian dilakukan pengecekan keseragaman warna pada suatu blok dengan beberapa *Error Measurement Methods*. Jika suatu blok sudah dianggap seragam atau blok lebih kecil daripada besar minimum blok setelah dibagi, maka blok tersebut tidak akan dibagi lagi, tetapi jika suatu blok belum dianggap cukup seragam, blok tersebut dibagi menjadi empat blok bagian. Proses ini disebut juga dengan proses **Divide** dalam algoritma Divide and Conquer. Pembagian blok ini direpresentasikan dalam bentuk Quadtree, suatu blok yang dapat dibagi akan menjadi sebuah *root* untuk empat simpul baru yaitu empat blok gambar hasil pembagian blok awal.

Setelah dilakukan proses divide, kita menuju ke proses **Conquer**. Tiap blok dalam tree yang tidak memiliki simpul daun (yang tidak dapat dibagi lagi) akan dinormalisasi warnanya dengan mencari rata-rata nilai red, green, dan blue pada blok tersebut dan merubah nilai RGB tiap piksel dengan nilai rata-rata RGB yang telah ditemukan.

Setelah semua blok tersebut selesai dinormalisasi, kita melakukan proses **Combine**. Proses ini dilakukan dengan menyatukan kembali setiap blok gambar yang sudah dinormalisasi menjadi suatu gambar yang utuh dengan dimensi yang sama dengan gambar semula. Hasil penggabungan semua blok ini adalah hasil akhir dari kompresi gambar dengan metode Quadtree ini.

BAB III

SOURCE CODE

1. main.cpp

```
#include "cli.hpp"
#include "blok.hpp"
#include "MAD.hpp"
#include "ImageProcess.hpp"
#include "gif.h"
#include <chrono>
#include <cstdlib>
#include <filesystem>
#include <iomanip>
using namespace std;
int main(int argc, char *argv[])
{
   system("clear");
   cli* handler = new cli();
   auto start = chrono::high_resolution_clock::now();
   ImageProcess image(handler->getImagePath());
   vector<vector<rgb>> rgbMatrix = image.getRGBMatrix();
   error_code ec;
   uintmax_t originalSize =
filesystem::file_size(handler->getImagePath(), ec);
   if (ec) {
       cerr << "Error getting input file size: " <<
```

```
ec.message() << endl;

   } else {

       cout << fixed << setprecision(2);

       cout << "Original image size: " << (originalSize /
1024.0) << " KB" << endl;

   }

   Blok* blok = new Blok(0, 0, rgbMatrix,
image.getWidth(), image.getHeight());

   if (blok->divide(handler->getmethod(),
handler->getRange(), handler->getMinBlok())) {

       cout << "Total blocks created: " <<
Blok::getCountBlok() << endl;

       cout << "Max depth reached: " <<
Blok::getMaxDepth() << endl;

       blok->normalizeRGB();

       GifWriter gifWriter;

       GifBegin(&gifWriter,
handler->getGifPath().c_str(), image.getWidth(),
image.getHeight(), 100);

       for (int level = 0; level <= Blok::getMaxDepth();
++level) {

           vector<vector<rgb>>
frameMatrix(image.getHeight(),
vector<rgb>(image.getWidth()));

           blok->reconstructGif(frameMatrix, level);

           uint8_t* buffer = new uint8_t[image.getWidth()
* image.getHeight() * 4];

           int idx = 0;

           for (int y = 0; y < image.getHeight(); ++y) {

               for (int x = 0; x < image.getWidth(); ++x)
{

                   buffer[idx++] =
```

```
frameMatrix[y][x].getRed();

                    buffer[idx++] =
frameMatrix[y][x].getGreen();

                    buffer[idx++] =
frameMatrix[y][x].getBlue();

                    buffer[idx++] = 255;

                }

            }

            GifWriteFrame(&gifWriter, buffer,
image.getWidth(), image.getHeight(), 100);

            delete[] buffer;

        }

        GifEnd(&gifWriter);

        vector<vector<rgb>>
outputMatrix(image.getHeight(),
vector<rgb>(image.getWidth()));

        blok->reconstructMatrix(outputMatrix);

        ImageProcess outputImage;

        outputImage.saveImage(handler->getOutputPath(),
outputMatrix);

        uintmax_t resultSize =
filesystem::file_size(handler->getOutputPath(), ec);

        if (ec) {

            cerr << "Error getting output file size: " <<
ec.message() << endl;

        } else {

            cout << "Result image size: " << (resultSize /
1024.0) << " KB" << endl;

        }

        cout << "Gif saved successfully: " <<
```

```
handler->getGifPath() << endl;

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

       auto duration =
chrono::duration_cast<chrono::milliseconds>(end - start);

       cout << "Processing time: " << duration.count() <<
" ms" << endl;

   }

   delete handler;

   delete blok;

   return 0;

}
```

2. cli.hpp

```
// cli.hpp

#ifndef CLI_HPP

#define CLI_HPP

#include <iostream>

#include <string>

using namespace std;

class cli

{

private:

   string imagePath;

   string outputPath;

   string gifPath;
```

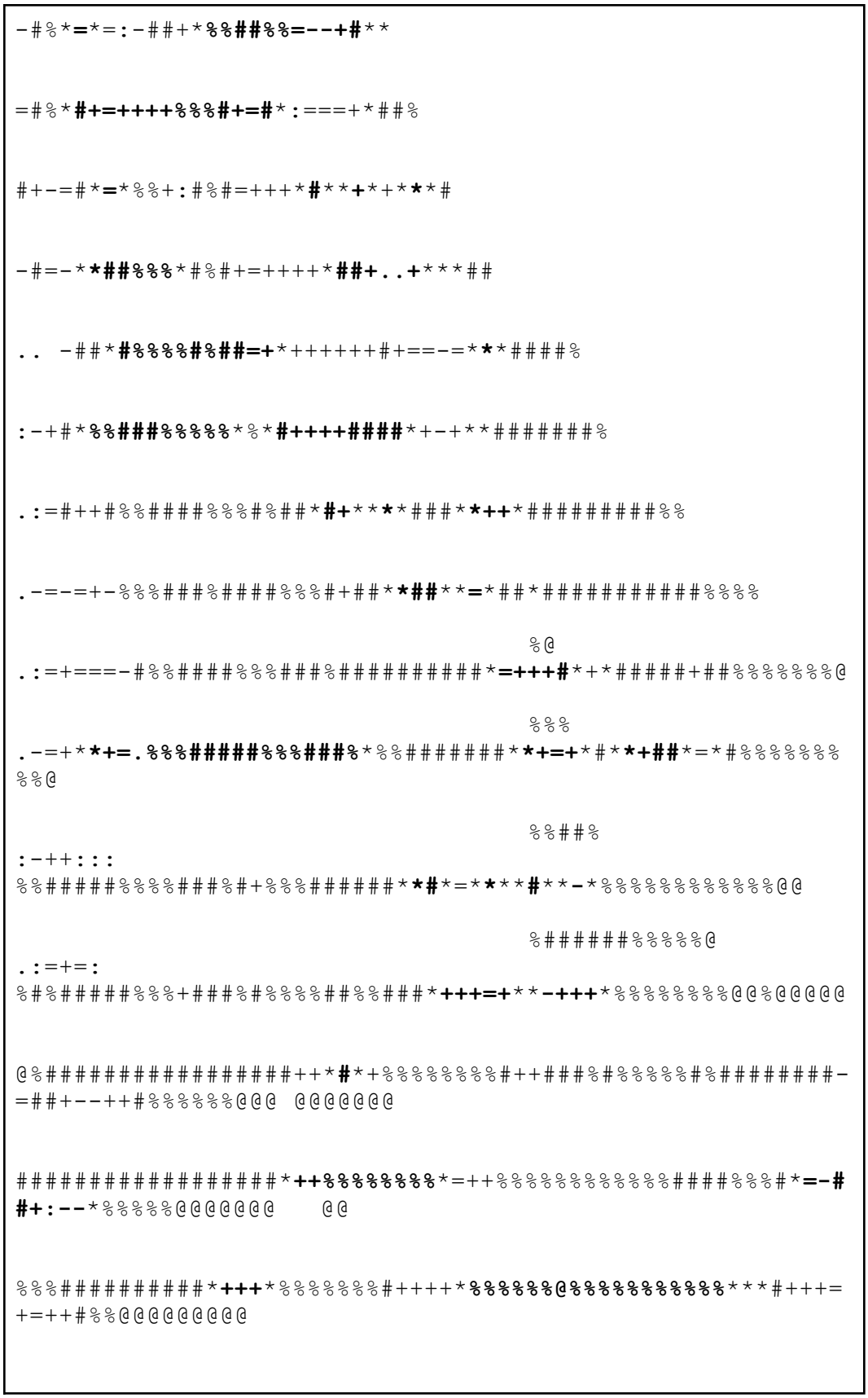
```
    int method;
    int range;
    int minBlok;
public:
    cli();
    ~cli();
    void setImagePath(string path);
    void setOutputPath(string path);
    void setMethod(int method);
    void setRange(int range);
    void setMinBlok(int minBlok);
    string getImagePath();
    string getGifPath();
    string getOutputPath();
    int getmethod();
    int getRange();
    int getMinBlok();
    void debugInfo();
    bool checkMethod();
    bool checkRange();
    bool checkMinBlok();
    void displayASCII();
};
#endif
// cli.hpp
```

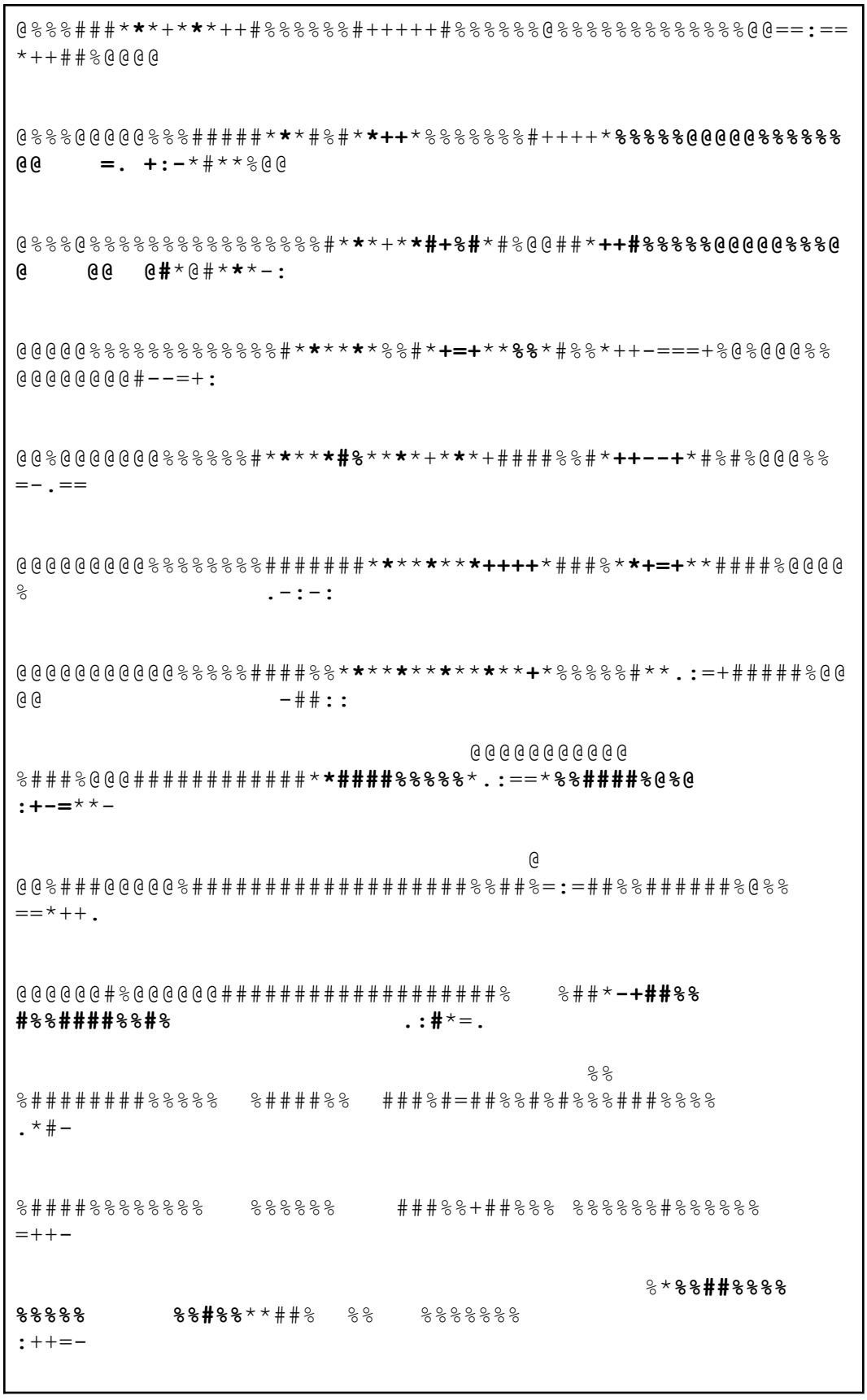
3. cli.cpp

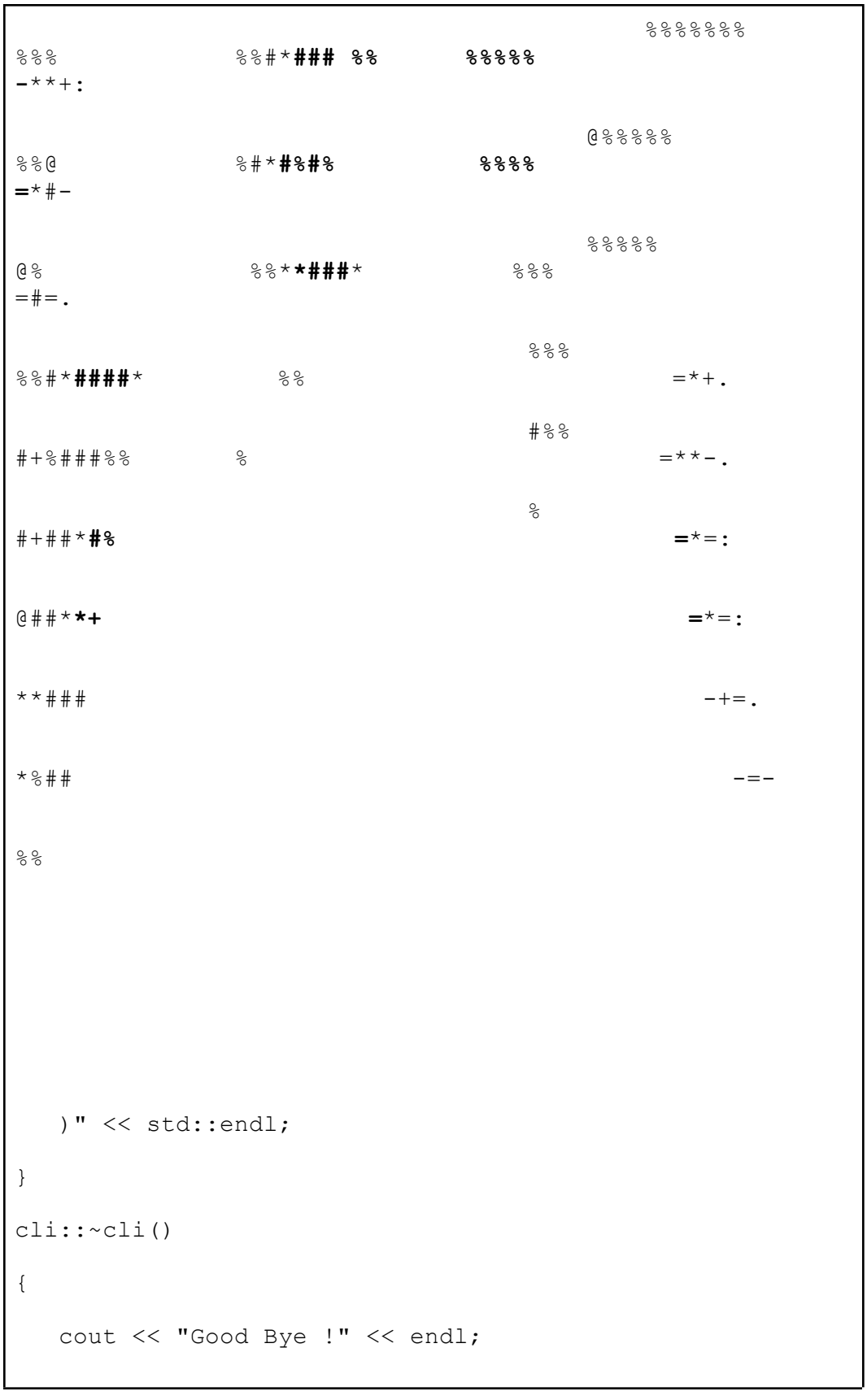
```
#include "cli.hpp"

cli::cli()

{

   cout << R"(

   .-----------------.  .-----------------.
.-----------------.  .-----------------.
.-----------------.  .-----------------.
.-----------------.

   | .---------------. || .---------------. ||
.--------------. || .--------------. || .--------------.
|| .---------------. || .---------------. |

   | | _____  _____ | || |  _________   | || |   _____
| || |      _______   | || |     ____     | || | ____
____ | || |  _________   | |

   | ||_   _||_   _|| || | |_   ___  |  | || |  |_   _|
| || |   .' ___  |  | || |   .'   `.   | || ||_   \  /
_|| || | |_   ___  |  | |

   | |   | | /\ | |   | || |   | |_  \_|  | || |    | |
| || |  / .'   \_|  | || |  /  .--.  \  | || |  |   \/
|   | || |   | |_  \_|  | |

   | |   | |/  \| |   | || |   |  _|  _   | || |    | |   _
| || |  | |          | || |  | |    | |  | || |  | |\  /|
|   | || |   |  _|  _   | |

   | |   |   /\   |  | || |  _| |___/ |  | || |   _| |__/
|  | || |  \ `.___.'\  | || |  \  `--'  /  | || | _|
|_\/_| |_ | || |  _| |___/ |  | |

   | |  |__/  \__|  | || | |_________|  | || |
|________|  | || |   `._____.'  | || |   `._____.'   | ||
||_____||_____|| || | |_________|  | |

   | |               | || |               | || |
| || |               | || |               | || |
| || |               | |

   | '--------------' || '--------------' ||
```

```
'--------------' || '--------------' || '--------------'
|| '--------------' || '--------------' |

    '-----------------'  '-----------------'
'-----------------'  '-----------------'
'-----------------'  '-----------------'
'-----------------'


   )" << endl;

   displayASCII();

   cout << "Please enter the image path: " << endl;

   cin >> imagePath;

   cout << "Please enter the output path: " << endl;

   cin >> outputPath;

   cout << "Please enter the GIF output path: " << endl;

   cin >> gifPath;

   do

   {

       cout << "Please enter the method: " << endl;

       cin >> method;

   } while (!checkMethod());

   do

   {

       cout << "Please enter the range: " << endl;

       cin >> range;

   } while (!checkRange());

   do

   {
```

```
        cout << "Please enter the minBlok: " << endl;
        cin >> minBlok;
    } while (!checkMinBlok());
}
void cli::displayASCII() {
    std::cout << R"(



**###

*******##%

%**######%%

%#########*%##

#*##+*+++###*#*%

#*#*=-+*#####*#*=+

#*+**#=***--#%%*---+*

##-*#*+++++*##%%#=-.-+

%#*++-.+#%###**#%#*-:==-+

@%#*+-:%@@@@%#:#%%%%++*=**#

:+#*+-=%#%%**==#%%%%*=+###
```





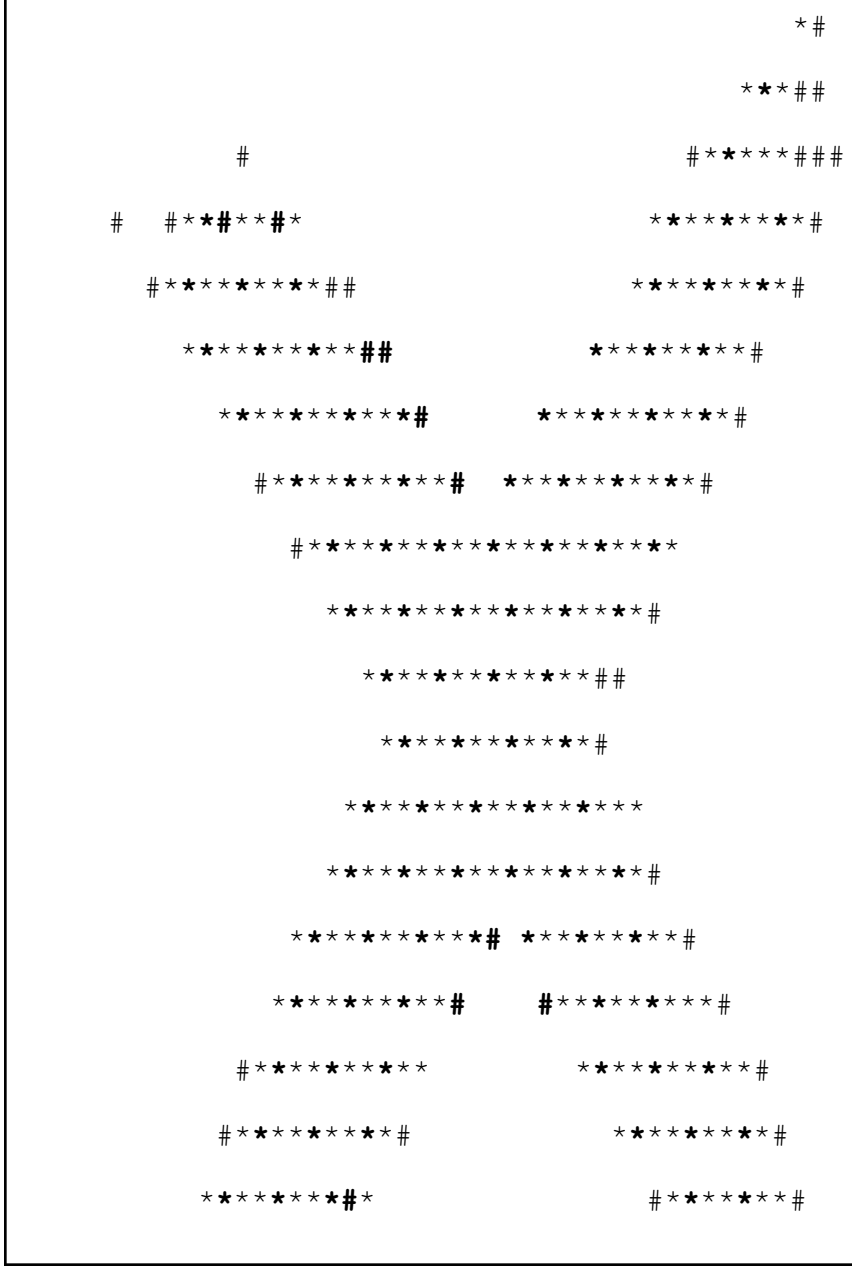


```
   )" << std::endl;
}
cli::~cli()
{
   cout << "Good Bye !" << endl;
```

```
}
void cli::setImagePath(string path)
{
   imagePath = path;
}
void cli::setOutputPath(string path)
{
   outputPath = path;
}
void cli::setMethod(int method)
{
   this->method = method;
}
void cli::setRange(int range)
{
   this->range = range;
}
void cli::setMinBlok(int minBlok)
{
   this->minBlok = minBlok;
}
string cli::getImagePath()
{
   return imagePath;
}
string cli::getOutputPath()
```

```
{
   return outputPath;
}
string cli::getGifPath()
{
   return gifPath;
}
int cli::getmethod()
{
   return method;
}
int cli::getRange()
{
   return range;
}
int cli::getMinBlok()
{
   return minBlok;
}
void cli::debugInfo()
{
   cout << "Image Path  : " << imagePath << endl;
   cout << "Output Path : " << outputPath << endl;
   cout << "Method      : " << method << endl;
   cout << "Range       : " << range << endl;
   cout << "Min Blok    : " << minBlok << endl;
```

```
}
bool cli::checkMethod()
{
   if (method < 1 || method > 5)
   {
       cout << R"(
```



```
          *******#                        **#***#

        *#******#

          ****

          ## #

         # #

          #


        )" << endl;

        cout << "Method is not valid ! (1 - 5)" << endl;


        return false;

    }

    return true;

}

bool cli::checkRange()

{

    return true;

}

bool cli::checkMinBlok()

{

    if (minBlok <= 0) {

        return false;

    }

    return true;

}
```

4. ImageProcess.hpp

```
#ifndef IMAGEPROCESS_HPP
#define IMAGEPROCESS_HPP
#include "stb_image.h"
#include "stb_image_write.h"
#include <string>
#include <iostream>
#include "rgb.hpp"
#include <vector>
using namespace std;
class ImageProcess {
   private:
        int width, height, channels;
        unsigned char* imageData;
   public:
        ImageProcess();
        ImageProcess(const string& filename);
        vector<vector<rgb>> getRGBMatrix();
        ~ImageProcess();
        int getWidth() const;
        int getHeight() const;
        void saveImage(const string& filename, const
```

```
vector<vector<rgb>>& rgbMatrix);

};

#endif
```

5. ImageProcess.cpp

```
#include "ImageProcess.hpp"

#include <iostream>

#define STB_IMAGE_IMPLEMENTATION

#define STB_IMAGE_WRITE_IMPLEMENTATION

#include "stb_image.h"

#include "stb_image_write.h"

ImageProcess::ImageProcess() : imageData(nullptr),
width(0), height(0), channels(0) {}

ImageProcess::ImageProcess(const string& filename) {

   imageData = stbi_load(filename.c_str(), &width,
&height, &channels, 0);

   if (imageData == nullptr) {

       cerr << "Error loading image: " <<
stbi_failure_reason() << endl;

       exit(1);

   }

}

vector<vector<rgb>> ImageProcess::getRGBMatrix() {

   vector<vector<rgb>> rgbMatrix(height,
```

```
vector<rgb>(width));

   for (int y = 0; y < height; y++) {

       for (int x = 0; x < width; x++) {

           int index = (y * width + x) * channels;

           rgbMatrix[y][x] = rgb(imageData[index],
imageData[index + 1], imageData[index + 2]);

       }

   }

   return rgbMatrix;

}

void ImageProcess::saveImage(const string& filename,
const vector<vector<rgb>>& rgbMatrix) {

   int width = rgbMatrix[0].size();

   int height = rgbMatrix.size();

   vector<unsigned char> imageData(width * height * 3);

   for (int y = 0; y < height; y++) {

       for (int x = 0; x < width; x++) {

           int index = (y * width + x) * 3;

           imageData[index] = rgbMatrix[y][x].getRed();

           imageData[index + 1] =
rgbMatrix[y][x].getGreen();

           imageData[index + 2] =
rgbMatrix[y][x].getBlue();

       }

   }

   if (stbi_write_png(filename.c_str(), width, height, 3,
imageData.data(), width * 3)) {

       cout << "Image saved successfully: " << filename
```

```
<< endl;

   } else {

       cerr << "Error saving image: " <<
stbi_failure_reason() << endl;

   }

}

int ImageProcess::getWidth() const {

   return width;

}

int ImageProcess::getHeight() const {

   return height;

}

ImageProcess::~ImageProcess() {

   if (imageData) {

       stbi_image_free(imageData);

   }

}
```

6. rgb.hpp

```
#ifndef RGB_HPP

#define RGB_HPP

#include <iostream>

#include <string>

using namespace std;
```

```
class rgb
{
private:
   int red;
   int green;
   int blue;
public:
   rgb();
   ~rgb();
   rgb(int r, int g, int b);
   rgb(const rgb &other);
   rgb& operator=(const rgb &other);
   rgb operator+(const rgb &other) const;
   rgb operator-(const rgb &other) const;
   rgb operator*(int n) const;
   rgb operator/(int n) const;
   int getRed() const;
   int getGreen() const;
   int getBlue() const;
   void setRed(int r);
   void setGreen(int g);
   void setBlue(int b);
};
#endif
```

7. rgb.cpp

```
#include "rgb.hpp"
rgb::rgb()
{
   red = 0;
   green = 0;
   blue = 0;
}
rgb::~rgb() {}
rgb::rgb(int r, int g, int b)
{
   red = r;
   green = g;
   blue = b;
}
rgb::rgb(const rgb &other)
{
   red = other.red;
   green = other.green;
   blue = other.blue;
}
```

```
rgb& rgb::operator=(const rgb &other)

{

   if (this != &other)

   {

       red = other.red;

       green = other.green;

       blue = other.blue;

   }

   return *this;

}

rgb rgb::operator+(const rgb &other) const

{

   return rgb(red + other.red, green + other.green, blue
+ other.blue);

}

rgb rgb::operator-(const rgb &other) const

{

   return rgb(red - other.red, green - other.green, blue
- other.blue);

}

rgb rgb::operator*(int n) const

{

   return rgb(red * n, green * n, blue * n);

}

rgb rgb::operator/(int n) const

{
```

```
   if (n == 0)
   {
       throw invalid_argument("Division by zero is not
allowed.");
   }
   return rgb(red / n, green / n, blue / n);
}
int rgb::getRed() const
{
   return red;
}
int rgb::getGreen() const
{
   return green;
}
int rgb::getBlue() const
{
   return blue;
}
void rgb::setRed(int r)
{
   red = r;
}
void rgb::setGreen(int g)
{
   green = g;
```

```
}
void rgb::setBlue(int b)
{
   blue = b;
}
```

8. blok.hpp

```
#ifndef BLOK_HPP
#define BLOK_HPP
#include <iostream>
#include <string>
#include <vector>
#include "rgb.hpp"
#include "MAD.hpp"
#include "Variance.hpp"
#include "Entropy.hpp"
#include "MPD.hpp"
using namespace std;
class Blok {
private:
   const int coordX;
   const int coordY;
   static int countBlok;
   static int maxDepth;
```

```
    const int sizeX;

    const int sizeY;

    bool isRoot;

    vector<vector<rgb>> image;

    int depth;

    Blok* children[4] = {nullptr, nullptr, nullptr,
nullptr};

public:

    Blok(int x, int y, vector<vector<rgb>> img, int sx,
int sy, int d = 0);

    Blok(const Blok& other);

    ~Blok();

    bool checkSeragam(int method, int range) const;

    bool divide(int method, int range, int minBlok);

    void normalizeRGB();

    void reconstructMatrix(vector<vector<rgb>>&
outputMatrix);

    void reconstructGif(vector<vector<rgb>>& outputMatrix,
int level);

    void printBlok(int level) const;



    static int getCountBlok();

    static int getMaxDepth();

};

#endif
```

9. blok.cpp

```
#include "blok.hpp"

int Blok::countBlok = 0;

int Blok::maxDepth = 0;

Blok::Blok(int x, int y, vector<vector<rgb>> img, int sx,
int sy, int d)

   : coordX(x), coordY(y), image(img), sizeX(sx),
sizeY(sy), depth(d), isRoot(true) {

   countBlok++;

}

Blok::Blok(const Blok& other)

   : coordX(other.coordX), coordY(other.coordY),
image(other.image),

     sizeX(other.sizeX), sizeY(other.sizeY),
depth(other.depth), isRoot(other.isRoot) {

   countBlok++;

}

Blok::~Blok() {

   for (int i = 0; i < 4; i++) {

       delete children[i];

   }

   countBlok--;

}

bool Blok::checkSeragam(int method, int range) const {

   if(method == 1) {

       Variance varianceCalc({});

       vector<double> variance =
varianceCalc.computeVarianceForMatrix(image);

       double var =
```

```
varianceCalc.calculateVariance(variance);

       if (var < range) {

           return true;

       }

   } else if(method == 2) {

       MAD madvalue({});

       vector<double> madval =
madvalue.computeMADForMatrix(image);

       double mad = madvalue.calculateMAD(madval);

       if (mad < range) {

           return true;

       }

   } else if(method == 3) {

       MPD mpdCalc(image);

       double mpd = mpdCalc.computeMPD();

       if (mpd < range) {

           return true;

       }

   } else if(method == 4) {

       Entropy entropy(image);

       double ent = entropy.calculateEntropy(sizeX,
sizeY);

       if (ent < range) {

           return true;

       }

   } else if(method == 5) {
```

```
        SSIM ssim(image);
        rgb meanColor = ssim.computeMeanColor(image);
        double ssimVal = ssim.computeSSIM(meanColor);
        // ssim.debugPrint();
        if (ssimVal > range) {
            return true;
        }
    } else {
        return false;
    }
    if (depth > maxDepth) {
        maxDepth = depth;
    }
    return false;
}
bool Blok::divide(int method, int range, int minBlok) {
    if (checkSeragam(method, range)) {
        isRoot = false;
        return false;
    }
    if (sizeX / 2 < minBlok || sizeY / 2 < minBlok) {
        isRoot = false;
        return false;
    }
    int halfX = sizeX / 2 + (sizeX % 2);
    int halfY = sizeY / 2 + (sizeY % 2);
```

```
   vector<vector<rgb>> img1(halfY, vector<rgb>(halfX));

   vector<vector<rgb>> img2(halfY, vector<rgb>(sizeX -
halfX));

   vector<vector<rgb>> img3(sizeY - halfY,
vector<rgb>(halfX));

   vector<vector<rgb>> img4(sizeY - halfY,
vector<rgb>(sizeX - halfX));

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

       for (int j = 0; j < halfX; j++) {

           img1[i][j] = image[i][j];

           if (j + halfX < sizeX) {

               img2[i][j] = image[i][j + halfX];

           }

       }

   }

   for (int i = 0; i < sizeY - halfY; i++) {

       for (int j = 0; j < halfX; j++) {

           img3[i][j] = image[i + halfY][j];

           if (j + halfX < sizeX) {

               img4[i][j] = image[i + halfY][j + halfX];

           }

       }

   }

   children[0] = new Blok(coordX, coordY, img1, halfX,
halfY, depth + 1);

   children[1] = new Blok(coordX + halfX, coordY, img2,
sizeX - halfX, halfY, depth + 1);

   children[2] = new Blok(coordX, coordY + halfY, img3,
```

```
halfX, sizeY - halfY, depth + 1);

   children[3] = new Blok(coordX + halfX, coordY + halfY,
img4, sizeX - halfX, sizeY - halfY, depth + 1);

   for (int i = 0; i < 4; i++) {

       children[i]->divide(method, range, minBlok);

   }

   return true;

}

void Blok::reconstructGif(vector<vector<rgb>>&
outputMatrix, int level) {

   if (depth > level) return;

   if (!isRoot || depth == level) {

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

           for (int j = 0; j < sizeX; j++) {

               outputMatrix[coordY + i][coordX + j] =
image[i][j];

           }

       }

   }

   for (int i = 0; i < 4; i++) {

       if (children[i] != nullptr) {

           children[i]->reconstructGif(outputMatrix,
level);

       }

   }

}

void Blok::normalizeRGB() {
```

```
        int totalRed = 0, totalGreen = 0, totalBlue = 0;
        int pixelCount = sizeX * sizeY;
        for (auto& row : image) {
            for (auto& pixel : row) {
                totalRed += pixel.getRed();
                totalGreen += pixel.getGreen();
                totalBlue += pixel.getBlue();
            }
        }
        int avgRed = totalRed / pixelCount;
        int avgGreen = totalGreen / pixelCount;
        int avgBlue = totalBlue / pixelCount;
        for (auto& row : image) {
            for (auto& pixel : row) {
                pixel.setRed(avgRed);
                pixel.setGreen(avgGreen);
                pixel.setBlue(avgBlue);
            }
        }

    for (int i = 0; i < 4; i++) {
        if (children[i] != nullptr) {
            children[i]->normalizeRGB();
        }
    }
}
```

```
void Blok::reconstructMatrix(vector<vector<rgb>>&
outputMatrix) {

   if (!isRoot) {

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

           for (int j = 0; j < sizeX; j++) {

               outputMatrix[coordY + i][coordX + j] =
image[i][j];

           }

       }

   }

   for (int i = 0; i < 4; i++) {

       if (children[i] != nullptr) {

           children[i]->reconstructMatrix(outputMatrix);

       }

   }

}

void Blok::printBlok(int level) const {

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

       cout << "  ";

   }

   cout << "|-- Blok at (" << coordX << ", " << coordY <<
"), size: "

        << sizeX << "x" << sizeY << ", depth: " << depth

        << (isRoot ? " [INTERNAL NODE]" : " [LEAF]") <<
endl;

   if (!isRoot) {

       cout << "RGB values at leaf node (" << coordX <<
", " << coordY << "): ";
```

```
        for (int i = 0; i < sizeY; i++) {
            for (int j = 0; j < sizeX; j++) {
                cout << "(" << image[i][j].getRed() << ",
"
                     << image[i][j].getGreen() << ", "
                     << image[i][j].getBlue() << ") ";
            }
            cout << endl;
        }
    }
    for (int i = 0; i < 4; i++) {
        if (children[i] != nullptr) {
            children[i]->printBlok(level + 1);
        }
    }
}
int Blok::getCountBlok() {
    return countBlok;
}
int Blok::getMaxDepth() {
    return maxDepth;
}
```

10. variance.hpp

```
#ifndef VARIANCE_HPP

#define VARIANCE_HPP


#include <vector>

#include <cmath>

#include <iostream>

#include "rgb.hpp"

using namespace std;


class Variance {

private:

    vector<double> data;


public:

    Variance(const vector<double>& inputData);

    double computeVariance();

    vector<double> computeVarianceForMatrix(const
vector<vector<rgb>>& rgbMatrix);

    double calculateMean() const;

    static double calculateVariance(vector<double>&
data);

    void printData() const;

};


#endif
```

11. variance.cpp

```
#include "variance.hpp"

Variance::Variance(const vector<double>& inputData) :
data(inputData) {}

double Variance::calculateMean() const {

   if (data.empty()) return 0.0;

   double sum = 0;

   for (double val : data) {

       sum += val;

   }

   return sum / data.size();

}

double Variance::computeVariance() {

   if (data.empty()) return 0.0;

   double mean = calculateMean();

   double variance = 0;

   for (double val : data) {

       variance += (val - mean) * (val - mean);

   }

   return variance / data.size();

}
```

```
vector<double>
Variance::computeVarianceForMatrix(const
vector<vector<rgb>>& rgbMatrix) {

   vector<double> variances;

   for (const auto& row : rgbMatrix) {

       vector<double> channelData;

       for (const auto& pixel : row) {

           channelData.push_back((pixel.getRed() +
pixel.getGreen() + pixel.getBlue()) / 3.0);

       }

       Variance var(channelData);

       variances.push_back(var.computeVariance());

   }

   return variances;

}

double Variance::calculateVariance(vector<double>&
data) {

   if (data.empty()) return 0.0;

   double mean = 0;

   for (double val : data) {

       mean += val;

   }

   mean /= data.size();
```

```
    double variance = 0;
    for (double val : data) {
        variance += (val - mean) * (val - mean);
    }
    return variance / data.size();
}
void Variance::printData() const {
    for (double val : data) {
        cout << val << " ";
    }
    cout << endl;
}
```

12. MAD.hpp

```
#ifndef MPD_HPP
#define MPD_HPP
#include <vector>
#include <algorithm>
#include "rgb.hpp"
using namespace std;
class MPD {
```

```
private:
   vector<vector<rgb>> data;
public:
   MPD(const vector<vector<rgb>>& imgData);
   double computeMPD() const;
   void printMPD();
};
#endif
```

13. MAD.cpp

```
#include "mad.hpp"
MAD::MAD(const std::vector<double>& inputData) :
data(inputData) {}
double MAD::calculateMean() const {
   double sum = 0;
   for (double value : data) {
       sum += value;
```

```
    }
    return sum / data.size();
}
double MAD::computeMAD() {
    double mean = calculateMean();
    double mad = 0.0;
    for (double value : data) {
        mad += abs(value - mean);
    }
    return mad / data.size();
}
vector<double> MAD::computeMADForMatrix(const
vector<vector<rgb>>& rgbMatrix) {
    vector<double> madValues(3, 0.0);
    vector<double> redValues;
    vector<double> greenValues;
    vector<double> blueValues;
    for (const auto& row : rgbMatrix) {
        for (const auto& pixel : row) {
            redValues.push_back(pixel.getRed());
            greenValues.push_back(pixel.getGreen());
            blueValues.push_back(pixel.getBlue());
        }
```

```
    }
    MAD redMAD(redValues);
    MAD greenMAD(greenValues);
    MAD blueMAD(blueValues);
    madValues[0] = redMAD.computeMAD();
    madValues[1] = greenMAD.computeMAD();
    madValues[2] = blueMAD.computeMAD();
    return madValues;
}
double MAD::calculateMAD(vector<double>& data) {
    double mad = 0.0;
    for (double value : data) {
        mad += abs(value);
    }
    return mad / data.size();
}
void MAD::printData() const {
    cout << "Data: ";
    for (double value : data) {
        cout << value << " ";
    }
    cout << endl;
}
```

14. MPD.hpp

```
#ifndef MPD_HPP

#define MPD_HPP

#include <vector>

#include <algorithm>

#include "rgb.hpp"

using namespace std;

class MPD {

private:

   vector<vector<rgb>> data;

public:

   MPD(const vector<vector<rgb>>& imgData);

   double computeMPD() const;

   void printMPD();

};

#endif
```

15. MPD.cpp

```
#include "MPD.hpp"

MPD::MPD(const vector<vector<rgb>>& inputData) :
data(inputData) {}
```

```
double MPD::computeMPD() const {
   if (data.empty() || data[0].empty()) return 0.0;
   int maxR = 0, maxG = 0, maxB = 0;
   int minR = 255, minG = 255, minB = 255;
   for (const auto& row : data) {
       for (const auto& pixel : row) {
           maxR = max(maxR, pixel.getRed());
           maxG = max(maxG, pixel.getGreen());
           maxB = max(maxB, pixel.getBlue());
           minR = min(minR, pixel.getRed());
           minG = min(minG, pixel.getGreen());
           minB = min(minB, pixel.getBlue());
       }
   }
   double D_R = maxR - minR;
   double D_G = maxG - minG;
   double D_B = maxB - minB;
   double D_RGB = (D_R + D_G + D_B) / 3.0;
   return D_RGB;
}
void MPD::printMPD() {
   cout << "MPD: " << computeMPD() << endl;
}
```

16. Entropy.hpp

```
#ifndef ENTROPY_HPP

#define ENTROPY_HPP

#include <iostream>

#include <vector>

#include <cmath>

#include "rgb.hpp"

using namespace std;

class Entropy {

   private:

        vector<vector<rgb>> imageData;

   public:

        Entropy();

        Entropy(const vector<vector<rgb>>& imgData);

        double calculateEntropy(int width, int height)
const;

};

#endif
```

17. Entropy.cpp

```
#include "Entropy.hpp"

Entropy::Entropy() : imageData() {}

Entropy::Entropy(const vector<vector<rgb>>& imgData)
: imageData(imgData) {}

double Entropy::calculateEntropy(int width, int
height) const {

   vector<int> histogramRed(256, 0),
histogramGreen(256, 0), histogramBlue(256, 0);

   double totalPixels = width * height;

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

       for(int j = 0; j < width; j++) {

           histogramRed[imageData[i][j].getRed()]++;


histogramGreen[imageData[i][j].getGreen()]++;


histogramBlue[imageData[i][j].getBlue()]++;

       }

   }

   double entropyRed = 0.0, entropyGreen = 0.0,
entropyBlue = 0.0;

   for(int i = 0; i < 256; i++) {

       if(histogramRed[i] > 0) {

           double p = histogramRed[i] / totalPixels;

           entropyRed -= p * log2(p);

       }
```

```
        if(histogramGreen[i] > 0) {

            double p = histogramGreen[i] /
totalPixels;

            entropyGreen -= p * log2(p);

        }

        if(histogramBlue[i] > 0) {

            double p = histogramBlue[i] / totalPixels;

            entropyBlue -= p * log2(p);

        }

    }

    return (entropyRed + entropyGreen + entropyBlue) /
3.0;

}
```

18. ssim.hpp

```
#ifndef SSIM_HPP

#define SSIM_HPP

#include <vector>

#include <cmath>

#include <iostream>

#include "rgb.hpp"

using namespace std;
```

```
class SSIM {

private:

   vector<vector<rgb>> data;

   vector<vector<unsigned char>> R;

   vector<vector<unsigned char>> G;

   vector<vector<unsigned char>> B;

   double C1 = 6.5025, C2 = 58.5225, WR = 0.299, WG =
0.587, WB = 0.114;

public:

   SSIM(const vector<vector<rgb>>& inputData) :
data(inputData) {

       int H = data.size();

       int W = data[0].size();

       R.resize(H, vector<unsigned char>(W));

       G.resize(H, vector<unsigned char>(W));

       B.resize(H, vector<unsigned char>(W));


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

           for (int j = 0; j < W; ++j) {

               R[i][j] = data[i][j].getRed();

               G[i][j] = data[i][j].getGreen();

               B[i][j] = data[i][j].getBlue();

           }
```

```
        }

    }

    double computeSSIMChannel(const
vector<vector<unsigned char>>& ori, unsigned char
compressedValue);

rgb computeMeanColor(const vector<vector<rgb>>&
image);

    double computeSSIM(const rgb& compressedColor);

    void debugPrint();

};

#endif
```

19. ssim.cpp

```
#include "ssim.hpp"

double SSIM::computeSSIMChannel(const
vector<vector<unsigned char>>& ori, unsigned char
compressedValue) {

    double meanOri = 0.0;

    double meanCmp =
static_cast<double>(compressedValue);

    double varOri = 0.0;
```

```
    double covar = 0.0;
    int count = 0;
    for (size_t i = 0; i < ori.size(); ++i) {
        for (size_t j = 0; j < ori[i].size(); ++j) {
            meanOri += ori[i][j];
            count++;
        }
    }
    meanOri /= count;
    for (size_t i = 0; i < ori.size(); ++i) {
        for (size_t j = 0; j < ori[i].size(); ++j) {
            varOri += pow(ori[i][j] - meanOri, 2);
            covar += (ori[i][j] - meanOri) *
(compressedValue - meanCmp);
        }
    }
    varOri /= count;
    covar /= count;
    double ssim = ((2 * meanOri * meanCmp + C1) * (2 *
covar + C2)) /
                  ((pow(meanOri, 2) + pow(meanCmp, 2)
+ C1) * (varOri + C2));
    return ssim;
}
```

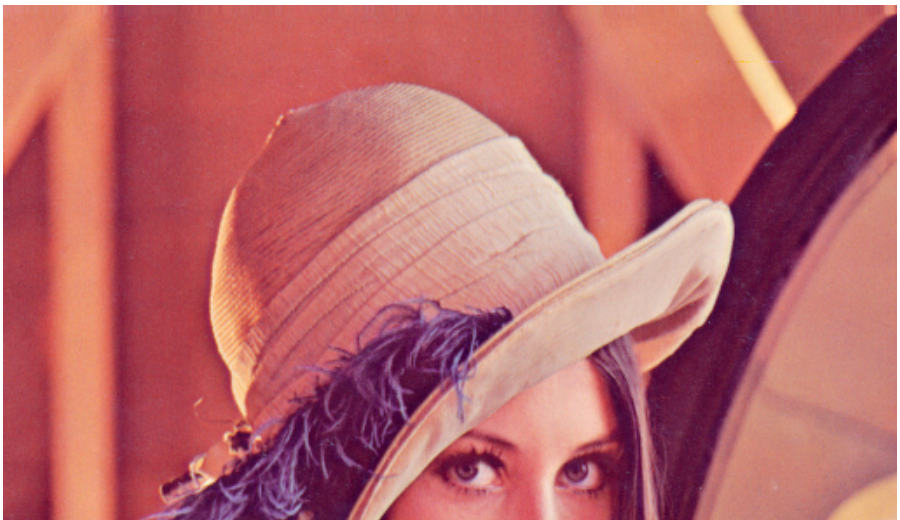
```
rgb SSIM::computeMeanColor(const vector<vector<rgb>>&
image) {

    long long sumR = 0, sumG = 0, sumB = 0;

    int H = image.size();

    int W = image[0].size();

    int total = H * W;


    for (const auto& row : image) {

        for (const auto& pixel : row) {

            sumR += pixel.getRed();

            sumG += pixel.getGreen();

            sumB += pixel.getBlue();

        }

    }


    return rgb(sumR / total, sumG / total, sumB /
total);

}

double SSIM::computeSSIM(const rgb& compressedColor)
{

   double ssimR = computeSSIMChannel(R,
compressedColor.getRed());

   double ssimG = computeSSIMChannel(G,
compressedColor.getGreen());
```

```
   double ssimB = computeSSIMChannel(B,
compressedColor.getBlue());

   return WR * ssimR + WG * ssimG + WB * ssimB;

}

void SSIM::debugPrint() {

   cout << "SSIM Debug Info:" << endl;

   cout << "C1: " << C1 << ", C2: " << C2 << endl;

   cout << "R channel:" << endl;

   for (const auto& row : R) {

       for (const auto& val : row) {

           cout << static_cast<int>(val) << " ";

       }

       cout << endl;

   }

   cout << "G channel:" << endl;

   for (const auto& row : G) {

       for (const auto& val : row) {

           cout << static_cast<int>(val) << " ";

       }

       cout << endl;

   }

   cout << "B channel:" << endl;

   for (const auto& row : B) {
```

```
        for (const auto& val : row) {
            cout << static_cast<int>(val) << " ";
        }
        cout << endl;
    }
}
```

BAB IV
PERCOBAAN

Percobaan digunakan dengan satu gambar dalam 5 metode berbeda (variance, MAD, MDP, entropy, SSIM) dengan threshold dan minBlok yang sama. Lalu dicoba dengan metode bebas dengan threshold dan minBlok yang berbeda. Gambar yang dipakai adalah berikut:



1. Variance
    - Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavarpng.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavargif.gif
Please enter the method:
1
Please enter the range:
10
Please enter the minBlok:
2
```

    - Output CLI

```
Original image size: 462.73 KB
Total blocks created: 58253
Max depth reached: 8
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavarpng.png
Result image size: 176.86 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavargif.gif
Processing time: 1311 ms
Good Bye !
```

- Output Gambar

- Output Gif

lennavargif.gif

2. Mean Absolute Deviation
   - Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamadpng.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamadgif.gif
Please enter the method:
2
Please enter the range:
10
Please enter the minBlok:
2
```

   - Output CLI

```
Original image size: 462.73 KB
Total blocks created: 17521
Max depth reached: 8
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamadpng.png
Result image size: 71.85 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamadgif.gif
Processing time: 1116 ms
Good Bye !
```

- Output Gambar

- Output GIF

lennamadgif.gif

3. Max Pixel Difference
   - Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamdppng.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamdpgif.gif
Please enter the method:
3
Please enter the range:
10
Please enter the minBlok:
2
```

   - Output CLI

```
Original image size: 462.73 KB
Total blocks created: 82493
Max depth reached: 8
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamdppng.png
Result image size: 221.51 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennamdpgif.gif
Processing time: 1272 ms
Good Bye !
```

- Output Gambar

- Output GIF

lennamdpgif.gif

4. Entropy
    - Input CLI

        Jika dimasukkan nilai threshold yang sama yaitu 10, maka gambar langsung dianggap seragam, sehingga diperlukan threshold yang lebih kecil yang dalam kasus ini adalah 5.

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennaentpng.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennaentgif.gif
Please enter the method:
4
Please enter the range:
5
Please enter the minBlok:
2
```

- Output CLI

```
Original image size: 462.73 KB
Total blocks created: 6181
Max depth reached: 6
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennaentpng.png
Result image size: 31.47 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennaentgif.gif
Processing time: 794 ms
Good Bye !
```

- Output Gambar

- Output GIF

  lennaentgif.gif

5. Structural Similiarity Index
   - Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennassimpng.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennassimgif.gif
Please enter the method:
5
Please enter the range:
10
Please enter the minBlok:
2
```

- Output CLI

```
Original image size: 462.73 KB
Total blocks created: 87381
Max depth reached: 8
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennassimpng.png
Result image size: 226.31 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennassimgif.gif
Processing time: 1725 ms
Good Bye !
```

- Output Gambar

- Output GIF

lennassimgif.gif

6. Variance (Threshold = 100, minBlok = 5)
    - Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavar100.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavar100gif.gif
Please enter the method:
1
Please enter the range:
100
Please enter the minBlok:
5
```

- Output CLI

```
Original image size: 462.73 KB
Total blocks created: 4041
Max depth reached: 6
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavar100.png
Result image size: 22.15 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lennavar100gif.gif
Processing time: 880 ms
Good Bye !
```

- Output Gambar

- Output GIF

  lennavar100gif.gif

7. Variance (Threshold = 15, minBlok = 20)

- Input CLI

```
Please enter the image path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna.png
Please enter the output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna15.png
Please enter the GIF output path:
/mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna15gif.gif
Please enter the method:
1
Please enter the range:
15
Please enter the minBlok:
20
```

- Output CLI

```
Original image size: 462.73 KB
Total blocks created: 341
Max depth reached: 4
Image saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna15.png
Result image size: 9.16 KB
Gif saved successfully: /mnt/c/Semester-4/Stima/Tucil/Tucil2_13523064_13523076/test/lenna15gif.gif
Processing time: 580 ms
Good Bye !
```

- Output Gambar

- Output GIF

  lenna15gif.gif

BAB V

ANALISIS PERCOBAAN

Untuk sebuah gambar N x N yang dipecah menjadi 4 sub-blok (quad-tree). secara kompleksitas waktu, worst case yang dapat terjadi adalah, setiap blok dianggap berbeda (tidak seragam). Sehingga akan dibagi terus sampai ukuran minimum (B x B).

Jumlah level maksimum rekursi adalah:

$$\log_4\left(\frac{N^2}{B^2}\right) = 2\log_2\left(\frac{N}{B}\right)$$

Jumlah blok atau simpul yang dibuat adalah :

$$O\left(\frac{N^2}{B^2}\right)$$

Maka kompleksitas waktu tetap linear dengan jumlah pixel yang akan diproses.

Kompleksitas waktu yang diperlukan adalah :

$$O\left(\frac{N^2}{B^2}\right) \times Kompleksitas\ Tiap\ Blok$$

Untuk Kompleksitas Ruang, penyimpanan gambar membutuhkan $O(N^2)$, Pohon yang dibuat maksimal membutuhkan $O(N^2)$. Artinya kompleksitas ruangnya adalah $O(N^2)$.

BAB VI
PENJELASAN BONUS

1. GIF

   Program mampu membuat output berupa file GIF dari proses kompresi yang dilakukan. GIF yang diharapkan adalah perubahan gambar yang terjadi *frame by frame* dari satu blok gambar dengan 1 warna, menjadi 4 blok gambar dengan 4 warna dan seterusnya sampai hasil kompresi tampak. Untuk mencapai ini, semua simpul yang ada pada Quadtree dinormalisasi terlebih dahulu (atau bisa membuat objek copy dari Quadtree yang ada kemudian dinormalisasi semua). Lalu dari Quadtree tersebut, dibentuk *frame by frame* dimulai dari depth 0. Iterasi dilakukan dengan membentuk frame dari blok di depth 0, kemudian semua blok di depth 1 dan seterusnya. Frame-frame ini akan disatukan dan membentuk sebuah animasi proses dalam bentuk file GIF.

2. SSIM

   Program mampu menggunakan metode SSIM untuk melakukan error checking tiap blok. Metode ini membandingkan kesamaan antara blok awal (original) dengan blok yang sudah di kompressi jika blok ini memang sudah seragam. Hasil perbandingan inilah yang dibandingkan dengan threshold, apakah sudah cukup seragam atau tidak.

LAMPIRAN

Link Repositori:https://github.com/Narrr21/Tucil2_13523064_13523076.git

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

Library stb_image:https://github.com/nothings/stb.git

Library gif:https://github.com/charlietangora/gif-h.git

Spesifikasi Tugas Kecil 2: Tucil2-Stima-2025.pdf