

Laporan Tugas Kecil Quadtree Image Compression

Tugas Kecil 2 - IF2211 Strategi Algoritma Semester II Tahun
2024/2025



Disusun oleh

Frederiko Eldad Mugiyono - 13523147

Daftar Isi

Daftar Isi.....	1
Daftar Gambar.....	2
1. Deskripsi Masalah.....	3
2. Analisis Landasan Teori.....	4
2.1. Algoritma Divide and Conquer.....	4
2.2. Implementasi Algoritma Divide and Conquer Dalam Quadtree.....	4
2.3. Implementasi Perhitungan Error.....	5
3. Source Code.....	7
3.1. Main.cpp.....	7
3.2. Quadtree.cpp.....	9
3.3. Variance.cpp.....	12
3.4. Meanabsolutedeviation.cpp.....	13
3.5. Maxpixeldifference.cpp.....	14
3.6. Entropy.cpp.....	15
3.7. Ascii.cpp.....	16
3.8. Imageloader.cpp.....	16
3.9. Iohandler.cpp.....	17
4. Uji Coba.....	20
4.1. Tangkapan Layar Uji Coba.....	20
4.2. Analisis Kompleksitas Algoritma.....	20
4.2.1. Kompleksitas Waktu.....	20
4.2.2. Kompleksitas Ruang.....	21
4.2.3. Kesimpulan.....	21
5. Pranala Repository.....	22
6. Lampiran.....	23

Daftar Tabel

Tabel 4.1.1. Tangkapan Layar Hasil Uji Coba 1.....	21
Tabel 4.1.2. Tangkapan Layar Hasil Uji Coba 2.....	22
Tabel 4.1.3. Tangkapan Layar Hasil Uji Coba 3.....	23
Tabel 4.1.4. Tangkapan Layar Hasil Uji Coba 4.....	24
Tabel 4.1.5. Tangkapan Layar Hasil Uji Coba 5.....	25
Tabel 4.1.6. Tangkapan Layar Hasil Uji Coba 6.....	27
Tabel 4.1.7. Tangkapan Layar Hasil Uji Coba 7.....	28

1. Deskripsi Masalah

Kompresi gambar merupakan proses penting dalam bidang pengolahan citra digital, khususnya untuk mengurangi ukuran file gambar tanpa menurunkan kualitas visual secara signifikan. Salah satu metode yang efektif untuk kompresi gambar adalah menggunakan struktur data Quadtree. Quadtree merupakan struktur data hierarkis yang bekerja dengan prinsip *divide and conquer*, di mana gambar secara rekursif dibagi menjadi empat bagian berdasarkan keseragaman warna atau intensitas pikselnya. Setiap bagian dari gambar tersebut kemudian diuji menggunakan parameter variansi untuk menentukan apakah masih perlu dilakukan pembagian lebih lanjut atau tidak.

Proses ini dimulai dengan gambar input yang diolah dalam bentuk matriks piksel RGB. Parameter seperti metode pengukuran variansi (Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy), threshold variansi, serta ukuran minimum blok piksel dapat dikonfigurasi pengguna sesuai kebutuhan. Algoritma akan menghitung variansi pada setiap blok dan membandingkannya dengan nilai threshold yang telah ditentukan. Apabila variansi blok berada di atas threshold dan ukuran blok masih di atas batas minimum, maka blok akan kembali dibagi menjadi empat bagian yang lebih kecil secara rekursif hingga mencapai kondisi tertentu. Blok-blok akhir yang tidak lagi dibagi akan dinormalisasi berdasarkan rata-rata warna atau intensitas pikselnya. Hasil akhir berupa gambar terkompresi direkonstruksi dari struktur Quadtree yang terbentuk, dan tingkat efisiensi kompresi dihitung sebagai persentase reduksi ukuran file dari gambar asli ke gambar hasil kompresi.

2. Analisis Landasan Teori

2.1. Algoritma *Divide and Conquer*

Algoritma *divide and conquer* bekerja dengan memecah masalah besar menjadi sub-masalah yang lebih kecil dan lebih mudah diselesaikan. Setelah sub-masalah diselesaikan, solusi-solusi tersebut digabungkan kembali untuk menghasilkan solusi akhir dari masalah awal. Dalam konteks kompresi gambar menggunakan Quadtree, algoritma ini efektif karena memungkinkan pemecahan masalah secara paralel dan efisien. Kompleksitas waktu algoritma divide and conquer dalam kasus ini bergantung pada tingkat rekursi serta jumlah piksel dalam gambar, yang secara umum menghasilkan kompleksitas logaritmik hingga linear-logaritmik tergantung pada parameter yang digunakan. Meskipun efisien, algoritma ini memerlukan penggunaan memori tambahan untuk menyimpan struktur rekursif selama proses pembagian gambar berlangsung.

2.2. Implementasi Algoritma *Divide and Conquer* Dalam Quadtree

Algoritma Divide and Conquer dalam implementasi Quadtree secara umum dapat dijelaskan sebagai berikut:

Pseudocode:

```
function subdivide(node, image):
    node.avgColor = calculateAverageColor(image, node.x, node.y,
    node.width, node.height)
    error = calculateError(image, node.x, node.y, node.width,
    node.height)

    if (node.width ≤ minBlockSize OR node.height ≤ minBlockSize
    OR error ≤ varThreshold):
        node.leaf = true
        return

    halfWidth = node.width / 2
    halfHeight = node.height / 2

    create 4 new nodes:
        top-left node: (node.x, node.y, halfWidth, halfHeight)
        top-right node: (node.x + halfWidth, node.y, node.width
        - halfWidth, halfHeight)
        bottom-left node: (node.x, node.y + halfHeight,
        halfWidth, node.height - halfHeight)
        bottom-right node: (node.x + halfWidth, node.y +
```

```

halfHeight, node.width - halfWidth, node.height - halfHeight)

    for each new node:
        subdivide(new node, image)

```

Penjelasan Algoritma:

1. Hitung rata-rata warna pada blok gambar saat ini.
2. Hitung nilai error berdasarkan metode yang dipilih pengguna.
3. Periksa apakah ukuran blok telah mencapai ukuran minimum atau apakah error sudah berada di bawah threshold. Jika ya, maka blok tersebut adalah daun (leaf).
4. Jika tidak memenuhi kondisi penghentian, blok akan dibagi menjadi empat bagian (sub-blok) yang lebih kecil.
5. Setiap sub-blok kemudian diproses secara rekursif dengan langkah-langkah yang sama hingga memenuhi kondisi penghentian.

Proses rekursi ini terus berlangsung hingga seluruh blok gambar memenuhi kondisi penghentian, menghasilkan struktur pohon yang mencerminkan distribusi keseragaman warna pada gambar. Struktur ini digunakan untuk merekonstruksi gambar terkompresi.

2.3. Implementasi Perhitungan Error

Berikut adalah implementasi metode perhitungan error yang digunakan dalam algoritma Quadtree:

Pseudocode Variance:

```

function calculateVariance(image, x, y, width, height):
    calculate sum and sum square of RGB values
    calculate mean of RGB
    calculate variance of RGB
    return average of variances (R, G, B)

```

Menghitung rata-rata dan variansi dari intensitas piksel dalam setiap blok gambar. Variansi menunjukkan tingkat keragaman intensitas warna dalam blok tersebut. Semakin tinggi variansi, semakin heterogen warna dalam blok.

Pseudocode Mean Absolute Deviation:

```

function calculateMAD(image, x, y, width, height):
    calculate mean RGB values
    calculate sum absolute deviation of RGB from mean
    return average of MAD values (R, G, B)

```

Menghitung rata-rata intensitas warna dalam blok, lalu menghitung rata-rata deviasi absolut setiap piksel terhadap rata-rata tersebut. Metode ini

menggambarkan seberapa jauh intensitas piksel dari nilai rata-rata warna blok tersebut.

Pseudocode Max Pixel Difference:

```
function calculateMPD(image, x, y, width, height):
    find maximum and minimum RGB values
    return average difference (max-min) of RGB
```

Menghitung selisih antara nilai piksel tertinggi dan terendah dalam satu blok. Metode ini menggambarkan rentang intensitas warna dalam blok gambar tersebut.

Pseudocode Entropy:

```
function calculateEntropy(image, x, y, width, height):
    calculate histogram of RGB
    calculate probability of each RGB value
    calculate entropy based on RGB probabilities
    return average entropy of RGB channels
```

Menghitung histogram intensitas piksel dalam blok untuk menentukan probabilitas setiap intensitas warna, lalu menghitung entropi untuk menggambarkan tingkat ketidakteraturan distribusi warna dalam blok tersebut. Semakin tinggi nilai entropi, semakin beragam warna dalam blok tersebut.

3. Source Code

3.1. Main.cpp

main.cpp

```
#include <chrono>
#include "quadtree/quadtree.hpp"
#include "iohandler/iohandler.hpp"
#include "imageloader/imageloader.hpp"
#include "ascii/ascii.hpp"

int main(){
    try{
        Ascii yuuka;
        yuuka.displayAscii();

        string imagePath = IOHandler::getImagePath("\nEnter the
image file path: ");
        int methodChoice = IOHandler::getMethodChoice();
        double varianceThreshold =
IOHandler::getVarianceThreshold();
        int minBlockSize = IOHandler::getMinBlockSize();
        string outputPath = IOHandler::getOutputPath();
        ErrorMethod* method =
IOHandler::chooseErrorMethod(methodChoice);
        FREE_IMAGE_FORMAT ext =
IOHandler::getImageFormat(outputPath);

        FIBITMAP* image = ImageLoader::loadImage(imagePath);

        QuadTree qTree;

        auto start = chrono::high_resolution_clock::now();
        qTree.buildTree(image, method, varianceThreshold,
minBlockSize);
        auto end = chrono::high_resolution_clock::now();

        chrono::duration<double> execTime = end - start;

        FIBITMAP* outputImg = nullptr;
        qTree.reconstructImg(outputImg);

        if (FreeImage_Save(ext, outputImg, outputPath.c_str(),
(ext == FIF_JPEG) ? 100 : 0)) {
            cout << "\nImage saved successfully at " <<
outputPath << "\n";
        } else {
    }
```

```

        throw runtime_error("Error: Failed to save the
image!");
    }

    auto originalSize = filesystem::file_size(imagePath);
    auto compressedSize = filesystem::file_size(outputPath);
    double compressionRatio = (1.0 - (double)compressedSize
/ originalSize) * 100;

    int depth = qTree.getDepth();
    int nodeCount = qTree.getNodesCount();

    cout <<
"=====\n";
    cout << "Compression Complete!\n";
    cout <<
"=====";
    cout << "Processing Time      : " << execTime.count() <<
" seconds\n";
    cout << "Original Size       : " << originalSize / 1024 << " KB\n";
    cout << "Compressed Size     : " << compressedSize / 1024 << " KB\n";
    cout << "Compression Percentage: " << compressionRatio << "%\n";
    cout << "Quadtree Depth      : " << depth << "\n";
    cout << "Total Nodes          : " << nodeCount << "\n";
    cout <<
"=====";
    cout <<

    delete method;
    FreeImage_Unload(image);
    FreeImage_Unload(outputImg);
    image = nullptr;
    outputImg = nullptr;
    FreeImage_DeInitialise();

} catch (const exception& e){
    cerr << "\nException: " << e.what() << "\n";
    return 1;
} catch (...){
    cerr << "\nUnknown error occurred!\n";
    return 1;
}

```

```
    return 0;
}
```

Merupakan titik awal eksekusi program. Di dalamnya, program membaca input dari pengguna, memuat gambar yang akan dikompresi, dan memanggil fungsi-fungsi utama untuk membangun Quadtree serta merekonstruksi gambar hasil kompresi.

3.2. Quadtree.cpp

quadtree.cpp

```
#include "quadtree/quadtree.hpp"

int QuadTree::Node::nodeCount = 0;
int QuadTree::Node::maxDepth = 0;

QuadTree::Node::Node(int x, int y, int width, int height, int
depth) : x(x), y(y), width(width), height(height), depth(depth){
    for (int i = 0; i < 4 ; ++i) children[i] = nullptr;
    if (depth > maxDepth) maxDepth = depth;
    ++nodeCount;
}

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

QuadTree::QuadTree() : root(nullptr), minBlockSize(0),
varThreshold(0.0), errorMethod(nullptr){}

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

void QuadTree::subdivide(Node* node, FIBITMAP* image){
    /* Calculate average color for this block */
    node->avgColor = calculateAverageColor(image, node->x,
node->y, node->width, node->height);

    /* Calculate error for this block */
    double error = errorMethod->calculateError(image, node->x,
node->y, node->width, node->height);

    /* Check whether pixel size has reached the minimum size or
error is less than threshold */
```

```

        if(node->width <= minBlockSize || node->height <=
minBlockSize || error <= varThreshold) {
            node->leaf = true;
            return;
        }

        int width = node->width;
        int height = node->height;
        int halfWidth = width / 2;
        int halfHeight = height / 2;

        /* Divide */
        node->children[0] = new Node(node->x, node->y, halfWidth,
halfHeight, node->depth + 1);
        node->children[1] = new Node(node->x + halfWidth, node->y,
node->width - halfWidth, halfHeight, node->depth + 1);
        node->children[2] = new Node(node->x, node->y + halfHeight,
halfWidth, node-> height - halfHeight, node->depth + 1);
        node->children[3] = new Node(node->x + halfWidth, node->y +
halfHeight, node->width - halfWidth, node->height - halfHeight,
node->depth + 1);

        /* Recursion */
        for (int i = 0; i < 4; ++i){
            subdivide(node->children[i], image);
        }
    }

RGBQUAD QuadTree::calculateAverageColor(FIBITMAP* image, int x,
int y, int width, int height) const{
    double sumR = 0, sumG = 0, sumB = 0;
    int pixelCount = width * height;

    for (int i = 0; i < height; ++i) {
        for (int j = 0; j < width; ++j) {
            RGBQUAD color;
            if (FreeImage_GetPixelColor(image, x + j, y + i,
&color)) {
                sumR += color.rgbRed;
                sumG += color.rgbGreen;
                sumB += color.rgbBlue;
            }
        }
    }

    RGBQUAD avgColor;
    avgColor.rgbRed = static_cast<BYTE>(sumR / pixelCount);
    avgColor.rgbGreen = static_cast<BYTE>(sumG / pixelCount);
    avgColor.rgbBlue = static_cast<BYTE>(sumB / pixelCount);
}

```

```

        avgColor.rgbBlue = static_cast<BYTE>(sumB / pixelCount);

        return avgColor;
    }

void QuadTree::buildTree(FIBITMAP* image, const ErrorMethod* method, double threshold, int minSize){
    errorMethod = method;
    varThreshold = threshold;
    minBlockSize = minSize;

    int width = FreeImage_GetWidth(image);
    int height = FreeImage_GetHeight(image);

    root = new Node(0, 0, width, height, 0);
    subdivide(root, image);
}

void QuadTree::reconstructNode(Node* node, FIBITMAP* image){
    if (!node) return;

    if (node->leaf){
        for(int i = 0; i < node->height; ++i){
            for(int j = 0; j < node->width; ++j){
                FreeImage_SetPixelColor(image, node->x + j,
node->y + i, &node->avgColor);
            }
        }
    } else{
        for (int i = 0; i < 4; ++i){
            reconstructNode(node->children[i], image);
        }
    }
}

void QuadTree::reconstructImg(FIBITMAP*& outputImg) {
    if (!root) {
        throw runtime_error("QuadTree root is null!");
    }

    if (root->width <= 0 || root->height <= 0) {
        throw runtime_error("Invalid image dimensions!");
    }

    outputImg = FreeImage_Allocate(root->width, root->height,
24);
    if (!outputImg) {
        throw runtime_error("Failed to allocate output image!");
    }
}

```

```

    }

    reconstructNode(root, outputImg);
}

int QuadTree::getDepth() const{
    return Node::maxDepth;
}

int QuadTree::getNodesCount() const{
    return Node::nodeCount;
}

```

Berisi implementasi kelas QuadTree beserta metode-metodenya. Metode utama dalam file ini meliputi:

1. subdivide(): Melakukan pembagian rekursif pada node hingga memenuhi kondisi tertentu.
2. calculateAverageColor(): Menghitung rata-rata warna dalam suatu blok.
3. buildTree(): Membangun struktur Quadtree berdasarkan gambar input.
4. reconstructImg(): Merekonstruksi gambar dari struktur Quadtree yang telah dibangun.

3.3. Variance.cpp

variance.cpp

```

#include "variance/variance.hpp"

double Variance::calculateError(FIBITMAP* image, int x, int y,
int width, int height) const {
    double sumR = 0, sumG = 0, sumB = 0, sumsqR = 0, sumsqG = 0,
sumsqB = 0, meanR = 0, meanG = 0, meanB = 0, varR = 0, varG = 0,
varB = 0;
    int pixels = width * height;

    for (int i = 0; i < height; ++i){
        for(int j = 0; j < width ; ++j){
            RGBQUAD color;
            if (FreeImage_GetPixelColor(image, x + j, y + i,
&color)){
                sumsqR += color.rgbRed * color.rgbRed, sumsqG +=
color.rgbGreen * color.rgbGreen, sumsqB += color.rgbBlue *
color.rgbBlue;
                sumR += color.rgbRed, sumG += color.rgbGreen,
sumB += color.rgbBlue;
            }
        }
    }
}

```

```

    }

    meanR = sumR / pixels, meanG = sumG / pixels, meanB = sumB / pixels;

    varR = sumsqR / pixels - meanR * meanR;
    varG = sumsqG / pixels - meanG * meanG;
    varB = sumsqB / pixels - meanB * meanB;

    /*  $\sigma_{RGB}^2 = (\sigma_R^2 + \sigma_G^2 + \sigma_B^2) / 3$  */
    return (varR + varG + varB) / 3.0;
}

```

Merupakan kelas turunan dari ErrorMethod yang mengimplementasikan metode perhitungan error menggunakan variansi. Fungsi calculateError() dalam file ini menghitung variansi dari nilai piksel dalam suatu blok untuk menentukan tingkat heterogenitas warna.

3.4. Meanabsolutedeviation.cpp

meanabsolutedeviation.cpp

```

#include "meanabsolutedeviation/meanabsolutedeviation.hpp"

double MAD::calculateError(FIBITMAP* image, int x, int y, int width, int height) const {
    double sumR = 0, sumG = 0, sumB = 0, meanR = 0, meanG = 0,
    meanB = 0, madR = 0, madG = 0, madB = 0;
    int pixels = width * height;
    RGBQUAD color;

    for (int i = 0; i < height; ++i){
        for(int j = 0; j < width ; ++j){
            if (FreeImage_GetPixelColor(image, x + j, y + i,
&color)){
                sumR += color.rgbRed, sumG += color.rgbGreen,
sumB += color.rgbBlue;
            }
        }
    }

    meanR = sumR / pixels, meanG = sumG / pixels, meanB = sumB / pixels;

    for (int i = 0; i < height; ++i){
        for(int j = 0; j < width ; ++j){
            if (FreeImage_GetPixelColor(image, x + j, y + i,

```

```

&color));
        madR += fabs(color.rgbRed - meanR), madG +=
fabs(color.rgbGreen - meanG), madB += fabs(color.rgbBlue -
meanB);
    }
}
}

madR /= pixels, madG /= pixels, madB /= pixels;

/* MAD_RGB = (MAD_R + MAD_G + MAD_B) / 3 */
return (madR + madG + madB) / 3.0;
}

```

Merupakan kelas turunan dari ErrorMethod yang mengimplementasikan metode Mean Absolute Deviation (MAD) untuk perhitungan error. Fungsi calculateError() menghitung rata-rata deviasi absolut dari nilai piksel terhadap rata-rata warna dalam blok.

3.5. Maxpixeldifference.cpp

maxpixeldifference.cpp

```

#include "maxpixeldifference/maxpixeldifference.hpp"

double MPD::calculateError(FIBITMAP* image, int x, int y, int
width, int height) const{
    double maxR = 0, maxG = 0, maxB = 0, minR = 255, minG = 255,
minB = 255;
    RGBQUAD color;

    for (int i = 0; i < height; ++i){
        for(int j = 0; j < width ; ++j){
            if (FreeImage_GetPixelColor(image, x + j, y + i,
&color)){
                maxR = max(maxR, (double)color.rgbRed), maxG =
max(maxG, (double)color.rgbGreen), maxB = max(maxB,
(double)color.rgbBlue);
                minR = min(minR, (double)color.rgbRed), minG =
min(minG, (double)color.rgbGreen), minB = min(minB,
(double)color.rgbBlue);
            }
        }
    }

/* D_RGB = (D_R + D_G + D_B) / 3 */
return (maxR - minR + maxG - minG + maxB - minB) / 3.0;
}

```

```
}
```

Merupakan kelas turunan ErrorMethod yang memiliki implementasi metode Max Pixel Difference (MPD). Fungsi calculateError() menghitung selisih antara nilai piksel maksimum dan minimum dalam suatu blok untuk menilai rentang perbedaan warna.

3.6. Entropy.cpp

entropy.cpp

```
#include "entropy/entropy.hpp"

double Entropy::calculateError(FIBITMAP* image, int x, int y,
int width, int height) const{
    vector<int> histR(256,0), histG(256,0), histB(256,0);
    double entropyR = 0, entropyG = 0, entropyB = 0, probR = 0,
probG = 0, probB = 0;
    int pixels = width * height;
    RGBQUAD color;

    for (int i = 0; i < height; ++i){
        for(int j = 0; j < width ; ++j){
            if (FreeImage_GetPixelColor(image, x + j, y + i,
&color)){
                ++histR[color.rgbRed], ++histG[color.rgbGreen],
++histB[color.rgbBlue];
            }
        }
    }

    for (int i = 0; i < 256; ++i){
        probR = (double)histR[i] / pixels;
        probG = (double)histG[i] / pixels;
        probB = (double)histB[i] / pixels;

        if (probR > 0) entropyR -= probR * log2(probR);
        if (probG > 0) entropyG -= probG * log2(probG);
        if (probB > 0) entropyB -= probB * log2(probB);
    }

    /* H_RGB = (H_R + H_G + H_B) / 3 */
    return (entropyR + entropyG + entropyB) / 3.0;
}
```

Merupakan kelas turunan dari ErrorMethod yang mengimplementasikan metode perhitungan error berdasarkan entropi. Fungsi calculateError() menghitung

entropi dari distribusi nilai piksel dalam blok untuk mengukur tingkat ketidakteraturan warna.

3.7. Ascii.cpp

ascii.cpp

```
#include "ascii/ascii.hpp"

Ascii::Ascii(){}
Ascii::~Ascii(){}

void Ascii::displayAscii() const{
    cout <<
"=====
=\n";
    cout << "                               YUUKAPRESS
\n";
    cout <<
"=====
=\n";
    cout << yuuka;
}
```

Merupakan kelas yang menampilkan *Ascii Art* dari file *headernya* sendiri melalui fungsi *displayAscii()*.

3.8. Imageloader.cpp

imageloader.cpp

```
#include "imageloader/imageloader.hpp"

FIBITMAP* ImageLoader::loadImage(const string& loadPath) {
    FreeImage_Initialise();

    FREE_IMAGE_FORMAT fif =
    FreeImage_GetFileType(loadPath.c_str(), 0);
    if (fif == FIF_UNKNOWN) {
        fif = FreeImage_GetFIFFFromFilename(loadPath.c_str());
    }

    if ((fif != FIF_UNKNOWN) &&
    FreeImage_FIFSupportsReading(fif)) {
        FIBITMAP* image = FreeImage_Load(fif, loadPath.c_str());
        if (!image) throw runtime_error("Failed to load image: "
+ loadPath);
```

```

        if (FreeImage_GetBPP(image) != 24) {
            FIBITMAP* temp = FreeImage_ConvertTo24Bits(image);
            FreeImage_Unload(image);
            image = temp;
        }

        return image;
    }

    throw runtime_error("Error: Unsupported image format " +
loadPath);
}

```

Merupakan kelas yang memiliki fungsi loadImage() untuk memuat gambar dan mengatasi apabila ada kesalahan dengan melemparkan *exception*.

3.9. Iohandler.cpp

iohandler.cpp

```

#include "iohandler/iohandler.hpp"

string IOHandler::getImagePath(const string& prompt){
    string path;
    cout << prompt;
    getline(cin, path);
    return path;
}

int IOHandler::getMethodChoice(){
    int choice;
    cout << "\nSelect an error calculation method:\n";
    cout << "1. Variance (0 - 65025)\n";
    cout << "2. Mean Absolute Deviation (0 - 255)\n";
    cout << "3. Max Pixel Difference (0 - 255)\n";
    cout << "4. Entropy (0 - 8)\n";
    cout << "Choice (1 - 4): ";
    cin >> choice;
    return (choice >= 1 && choice <= 4) ? choice : 1;
}

double IOHandler::getVarianceThreshold(){
    double threshold;
    cout << "\nEnter threshold value: ";
    cin >> threshold;
    return threshold;
}

```

```

int IOHandler::getMinBlockSize(){
    int blockSize;
    cout << "\nEnter minimum block size: ";
    cin >> blockSize;
    return blockSize;
}

float IOHandler::getTargetRatio(){
    float ratio;
    cout << "\nEnter target compression ratio (0.0 - 100.0): ";
    cin >> ratio;
    return (ratio >= 0.0f && ratio <= 100.0f) ? ratio : 0.0f;
}

string IOHandler::getOutputPath(){
    string path;
    cout << "\nEnter the output image path: ";
    cin.ignore(numeric_limits<streamsize>::max(), '\n');
    getline(cin, path);
    return path;
}

ErrorMethod* IOHandler::chooseErrorMethod(int choice){
    switch (choice){
        case 1: return new Variance();
        case 2: return new MAD();
        case 3: return new MPD();
        case 4: return new Entropy();
        default: return nullptr;
    }
}

FREE_IMAGE_FORMAT IOHandler::getImageFormat(const string& filename){
    string ext =
filesystem::path(filename).extension().string();

    transform(ext.begin(), ext.end(), ext.begin(), ::tolower);

    if (ext == ".png") return FIF_PNG;
    if (ext == ".jpg" || ext == ".jpeg") return FIF_JPEG;
    if (ext == ".bmp") return FIF_BMP;
    if (ext == ".tiff") return FIF_TIFF;
    if (ext == ".gif") return FIF_GIF;
    if (ext == ".tga") return FIF_TARGA;

    return FIF_UNKNOWN;
}

```

```
}
```

Merupakan kelas yang mengatasi permasalahan input dan output. Memiliki fungsi-fungsi yang digunakan dalam keberjalanan program saat memasukkan gambar, dan atribut-atribut yang digunakan dalam kompresi hingga output gambar yang sudah dikompresi.

4. Uji Coba

Program diuji dengan menggunakan masukan berupa gambar.

4.1. Tangkapan Layar Uji Coba

```
Enter the image file path: test/1.png
Select an error calculation method:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Choice (1-4): 1

Enter threshold value: 16

Enter minimum block size: 2

Enter the output image path: test/1c.png
Image saved successfully at test/1c.png

=====
Compression Complete!
=====
Processing Time      : 1.20618 seconds
Original Size       : 1513 KB
Compressed Size     : 406 KB
Compression Percentage: 73.1808%
Quadtree Depth      : 10
Total Nodes          : 150705
=====

Tucil2_13523147 on ✵ main [$x?] took 15s
> |
```



The image shows two side-by-side anime-style illustrations of a character with purple hair and a halo, wearing a white lab coat over a blue and white striped shirt. In both images, the character is holding a bowl of ramen with chopsticks. The character's expression is slightly surprised or excited. The background is white with some light gray radial patterns. The left image is labeled "Input" and the right image is labeled "Output".

Tabel 4.1.1. Tangkapan Layar Hasil Uji Coba 1

```
Enter the image file path: test/2.jpg  
Select an error calculation method:  
1. Variance  
2. Mean Absolute Deviation  
3. Max Pixel Difference  
4. Entropy  
Choice (1-4): 4  
  
Enter threshold value: 4  
  
Enter minimum block size: 32  
  
Enter the output image path: test/2c.jpg  
  
Image saved successfully at test/2c.jpg  
  
=====  
Compression Complete!  
=====
```

Processing Time : 3.89348 seconds
Original Size : 1122 KB
Compressed Size : 756 KB
Compression Percentage: 32.6205%
Quadtree Depth : 6
Total Nodes : 4969



Input



Output

Tabel 4.1.2. Tangkapan Layar Hasil Uji Coba 2

```
Enter the image file path: test/3.jpg
Select an error calculation method:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Choice (1-4): 2

Enter threshold value: 8

Enter minimum block size: 2

Enter the output image path: test/3c.jpg
Image saved successfully at test/3c.jpg

-----
Compression Complete!
-----
Processing Time      : 1.6186 seconds
Original Size        : 513 KB
Compressed Size      : 484 KB
Compression Percentage: 5.59949%
Quadtree Depth       : 10
Total Nodes          : 61929
-----
```



The image shows two side-by-side anime-style illustrations of a character with long purple hair and blue eyes, wearing a light blue blouse and a black skirt. In the left image, labeled 'Input', the character is standing with one leg slightly bent. In the right image, labeled 'Output', the character appears slightly more compressed or pixelated, particularly in the background and the edges of her clothing.

Tabel 4.1.3. Tangkapan Layar Hasil Uji Coba 3

```
Enter the image file path: test/4.png
Select an error calculation method:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Choice (1-4): 3

Enter threshold value: 16

Enter minimum block size: 4

Enter the output image path: test/4c.png
Image saved successfully at test/4c.png

=====
Compression Complete!
=====
Processing Time      : 9.98472 seconds
Original Size        : 4506 KB
Compressed Size      : 629 KB
Compression Percentage: 86.0245%
Quadtree Depth       : 10
Total Nodes          : 206845
=====


```

Input	Output
-------	--------

Tabel 4.1.4. Tangkapan Layar Hasil Uji Coba 4

<pre> Enter the image file path: test/5.jpg Select an error calculation method: 1. Variance 2. Mean Absolute Deviation 3. Max Pixel Difference 4. Entropy Choice (1-4): 1 Enter threshold value: 16 Enter minimum block size: 8 Enter the output image path: test/5c.jpg Image saved successfully at test/5c.jpg ===== Compression Complete! ===== Processing Time : 9.32238 seconds Original Size : 2676 KB Compressed Size : 2471 KB Compression Percentage: 7.68142% Quadtree Depth : 9 Total Nodes : 108865 </pre>	 Input	 Output
--	---	---

Tabel 4.1.5. Tangkapan Layar Hasil Uji Coba 5

```
Enter the image file path: test/6.png
Select an error calculation method:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Choice (1-4): 1

Enter threshold value: 32

Enter minimum block size: 8

Enter the output image path: test/6c.png
Image saved successfully at test/6c.png

=====
Compression Complete!
=====
Processing Time      : 4.22026 seconds
Original Size        : 7022 KB
Compressed Size      : 231 KB
Compression Percentage: 96.7038%
Quadtree Depth       : 9
Total Nodes          : 65809
```



The image shows two side-by-side anime-style illustrations of a character with purple hair and a blue outfit. The character has her hands behind her head and is wearing a small halo. In the bottom left corner of each illustration, there is a small glowing blue cube with a circular logo on it. The character's body is mostly blue, with a white belt and white trim on her outfit. The background is plain white.

Input

Output

Tabel 4.1.6. Tangkapan Layar Hasil Uji Coba 6

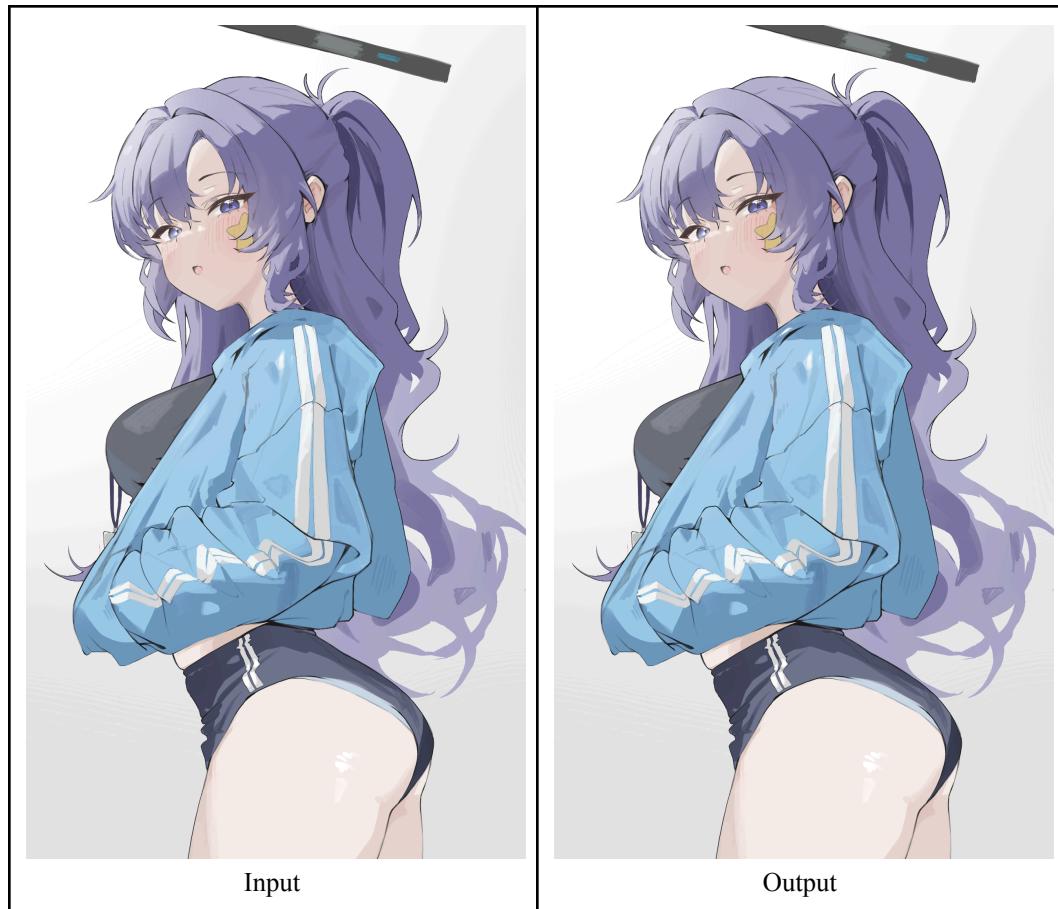
```
Enter the image file path: test/7.png
Select an error calculation method:
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
Choice (1-4): 1

Enter threshold value: 32

Enter minimum block size: 4

Enter the output image path: test/7c.png
Image saved successfully at test/7c.png

=====
Compression Complete!
=====
Processing Time      : 12.1177 seconds
Original Size       : 6271 KB
Compressed Size     : 538 KB
Compression Percentage: 91.4178%
Quadtree Depth      : 10
Total Nodes          : 163685
=====
```



Tabel 4.1.7. Tangkapan Layar Hasil Uji Coba 7

4.2. Analisis Kompleksitas Algoritma

4.2.1. Kompleksitas Waktu

Proses subdivide (pembagian): Algoritma Quadtree membagi gambar menjadi empat blok secara rekursif. Pada kasus terburuk, pembagian ini terus terjadi sampai ukuran blok mencapai ukuran minimum yang diatur pengguna. Jika ukuran gambar adalah $N \times N$ piksel, maka kedalaman maksimum pohon kira-kira sebesar $\log(N)$.

Perhitungan warna rata-rata dan error: Untuk setiap blok, algoritma menghitung nilai rata-rata warna dengan kompleksitas $O(w \times h)$, dengan w dan h adalah lebar dan tinggi blok tersebut. Namun, secara keseluruhan, setiap piksel hanya diproses sejumlah konstan kali selama algoritma dijalankan. Oleh karena itu, kompleksitas total untuk menghitung rata-rata warna dan error pada seluruh gambar adalah $O(N^2)$.

Total kompleksitas waktu: Berdasarkan analisis di atas, kompleksitas waktu keseluruhan algoritma adalah $O(N^2)$, dengan N^2 merupakan total jumlah piksel pada gambar.

4.2.2. Kompleksitas Ruang

Algoritma menggunakan struktur data berupa pohon Quadtree. Pada kasus terburuk, struktur pohon akan menyimpan sebanyak jumlah maksimum blok yang bisa dihasilkan dengan ukuran blok terkecil $s \times s$ piksel.

Dengan demikian, jumlah maksimum simpul yang disimpan dalam struktur pohon adalah sekitar $O(N^2 / s^2)$.

4.2.3. Kesimpulan

Kompleksitas waktu algoritma adalah $O(N^2)$, dengan N merupakan ukuran dimensi gambar.

Kompleksitas ruang algoritma adalah $O(N^2 / s^2)$, tergantung ukuran minimum blok yang diatur pengguna.

Kinerja algoritma akan dipengaruhi oleh parameter seperti ukuran gambar, ukuran blok minimum, dan nilai threshold variansi yang dipilih oleh pengguna.

5. Pranala Repository

Tucil2_13523147

6. Lampiran

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	✓	