

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma
Kompresi Gambar Dengan Metode Quadtree berbasis *Divide and Conquer*
Semester II Tahun 2024/2025



disusun oleh:

Boye Mangaratua Ginting (13523127)

Ivant Samuel Silaban (13523129)

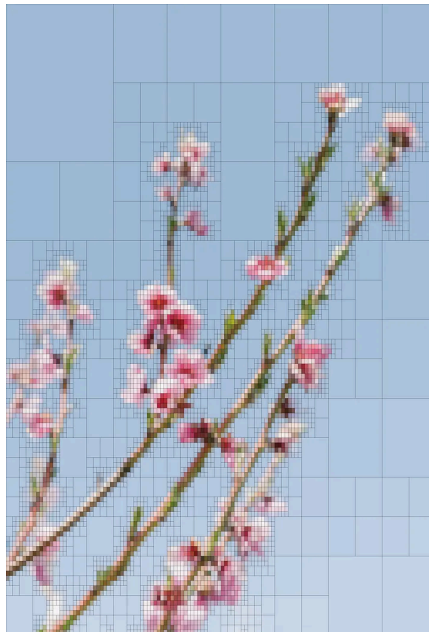
INSTITUT TEKNOLOGI BANDUNG

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.

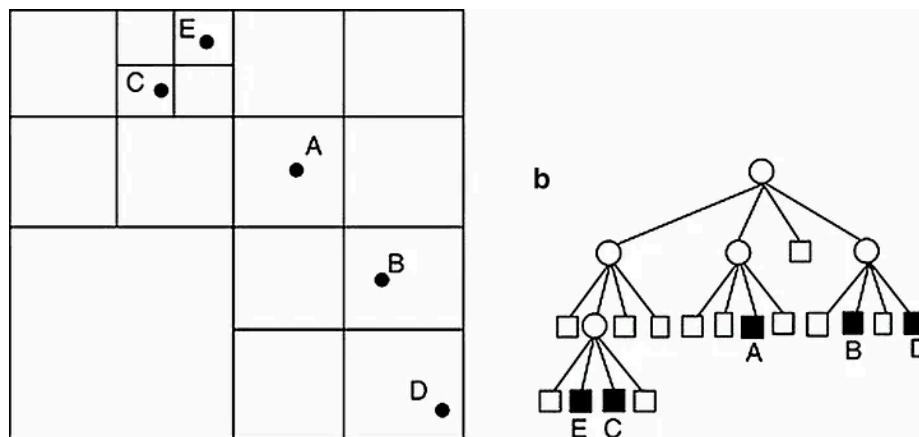
BAB II

Analisis Algoritma

2.1 Prinsip Dasar Quadtree

Quadtree adalah struktur data hirarki yang membagi ruang 2D menjadi empat kuadran. Prinsip dasarnya adalah:

1. Mulai dengan membuat seluruh gambar sebagai satu node.
2. Evaluasi apakah node perlu dibagi berdasarkan metrik error tertentu.
3. Jika error melebihi threshold, bagi node menjadi empat anak node (kuadran).
4. Lakukan langkah 2-3 secara rekursif untuk setiap anak node.
5. Jika error di bawah threshold atau ukuran mencapai minimum, simpan node sebagai leaf node dengan nilai warna rata-rata



2.2 Struktur Data

Program ini menggunakan bahasa C++, dengan struktur data sebagai berikut:

```
struct Pixel {
    unsigned char r, g, b;
};

struct BlockStats {
    double meanR, meanG, meanB;
    double varianceR, varianceG, varianceB;
    double madR, madG, madB;
    double maxDiffR, maxDiffG, maxDiffB;
    double entropyR, entropyG, entropyB;
};

class QuadTreeNode {
public:
    int x, y, size;
    bool isLeaf;
```

```

    Pixel avgColor;
    QuadTreeNode* children[4];

    QuadTreeNode(int x, int y, int size);
    ~QuadTreeNode();
};

```

2.3 Metode Perhitungan Error.

Terdapat empat metode perhitungan error:

2.3.1 Variance (Varians)

Mengukur seberapa jauh nilai pixel menyimpang dari nilai rata-rata. Variance mengukur seberapa bervariasi nilai pixel dalam suatu area dengan banyak detail atau perubahan warna.

- Rentang yang disarankan: 10.0 - 100.0
- Rentang kerja umum: 20.0 - 50.0
- Nilai rendah (<20): Kompresi ringan, kualitas tinggi.
- Nilai tinggi (>50): Kompresi agresif, mungkin akan memiliki artefak visual.

Formula:

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

Implementasi:

```

double calculateVariance(vector<vector<Pixel>>& data, int x,
int y, int size, Pixel avgColor) {
    double varR = 0.0, varG = 0.0, varB = 0.0;
    int count = 0;

    for (int j = y; j < y + size && j < data.size(); j++) {
        for (int i = x; i < x + size && i < data[j].size();
i++) {
            double dr = data[j][i].r - avgColor.r;

```

```

        double dg = data[j][i].g - avgColor.g;
        double db = data[j][i].b - avgColor.b;

        varR += dr * dr;
        varG += dg * dg;
        varB += db * db;
        count++;
    }
}

if (count > 0) {
    varR /= count;
    varG /= count;
    varB /= count;
}

return (varR + varG + varB) / 3.0;
}

```

2.3.2 Mean Absolute Deviation (MAD)

Mengukur rata-rata selisih absolut nilai pixel dengan nilai rata-rata. MAD cenderung memberikan nilai yang lebih kecil dibandingkan variance karena mengukur deviasi rata-rata, bukan selisih kuadrat.

- Rentang yang disarankan: 5.0 - 50.0
- Rentang kerja umum: 10.0 - 25.0
- Nilai rendah (<10): Kompresi ringan, preservasi detail.
- Nilai tinggi (>50): Kompresi tinggi, detail halus hilang..

Formula:

$MAD_c = \frac{1}{N} \sum_{i=1}^N P_{i,c} - \mu_c $	
$MAD_{RGB} = \frac{MAD_R + MAD_G + MAD_B}{3}$	
MAD_c	= Mean Absolute Deviation tiap kanal warna c (R, G, B) dalam satu blok
$P_{i,c}$	= Nilai piksel pada posisi i untuk kanal warna c
μ_c	= Nilai rata-rata tiap piksel dalam satu blok
N	= Banyaknya piksel dalam satu blok

Implementasi:

```

double calculateMAD(vector<vector<Pixel>>& data, int x, int y,
int size, Pixel avgColor) {
    double madR = 0.0, madG = 0.0, madB = 0.0;

```

```

int count = 0;

for (int j = y; j < y + size && j < data.size(); j++) {
    for (int i = x; i < x + size && i < data[j].size();
i++) {
        madR += abs(data[j][i].r - avgColor.r);
        madG += abs(data[j][i].g - avgColor.g);
        madB += abs(data[j][i].b - avgColor.b);
        count++;
    }
}

if (count > 0) {
    madR /= count;
    madG /= count;
    madB /= count;
}

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

```

2.3.3 Max Pixel Difference

Mengukur selisih maksimum antara nilai pixel dengan nilai rata-rata. Metode ini sensitif terhadap outlier karena hanya mempertimbangkan selisih pixel maksimum.

- Rentang yang disarankan: 15.0 - 100.0
- Rentang kerja umum: 30.0 - 60.0
- Nilai rendah (<20): Sedikit kompresi, hampir tidak ada loss.
- Nilai tinggi (>50): Kompresi signifikan, area dengan kontras tinggi dapat terdistorsi.

Formula:

$D_c = \max(P_{i,c}) - \min(P_{i,c})$	
$D_{RGB} = \frac{D_R + D_G + D_B}{3}$	
D_c	= Selisih antara piksel dengan nilai max dan min tiap kanal warna c (R, G, B) dalam satu blok
$P_{i,c}$	= Nilai piksel pada posisi i untuk channel warna c

Implementasi:

```

double calculateMaxDifference(vector<vector<Pixel>>& data,
int x, int y, int size) {
    unsigned char minR = 255, minG = 255, minB = 255;
    unsigned char maxR = 0, maxG = 0, maxB = 0;

    for (int j = y; j < y + size && j < data.size(); j++) {
        for (int i = x; i < x + size && i < data[j].size();
i++) {

```

```

        minR = min(minR, data[j][i].r);
        minG = min(minG, data[j][i].g);
        minB = min(minB, data[j][i].b);

        maxR = max(maxR, data[j][i].r);
        maxG = max(maxG, data[j][i].g);
        maxB = max(maxB, data[j][i].b);
    }
}

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

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

```

2.3.4 Entropy

Mengukur ketidakpastian atau randomness dalam data. Entropy memiliki rentang nilai yang lebih kecil karena mengukur randomness/kompleksitas data dalam bit. Nilai threshold kecil saja sudah dapat memberikan efek kompresi yang signifikan.

- Rentang yang disarankan: 0.3-2.0
- Rentang kerja umum: 0.5-1.2
- Nilai rendah (<0.5): Kompresi ringan
- Nilai tinggi (>1.2): Kompresi agresif

Formula:

$H_c = - \sum_{i=1}^N P_c(i) \log_2(P_c(i))$	
$H_{RGB} = \frac{H_R + H_G + H_B}{3}$	
H_c	= Nilai entropi tiap kanal warna c (R, G, B) dalam satu blok
$P_c(i)$	= Probabilitas piksel dengan nilai i dalam satu blok untuk tiap kanal warna c (R, G, B)

Implementasi:

```

double calculateEntropy(vector<vector<Pixel>>& data, int x, int
y, int size) {
    // Hitung histogram untuk setiap channel
    const int BINS = 256;
    vector<int> histR(BINS, 0), histG(BINS, 0), histB(BINS, 0);
    int totalPixels = 0;

```



```

    for (int j = y; j < y + size && j < data.size(); j++) {
        for (int i = x; i < x + size && i < data[j].size(); i++)
        {
            histR[data[j][i].r]++;
            histG[data[j][i].g]++;
            histB[data[j][i].b]++;
            totalPixels++;
        }
    }

    double entropyR = 0.0, entropyG = 0.0, entropyB = 0.0;
    for (int i = 0; i < BINS; i++) {
        if (histR[i] > 0) {
            double pR = (double)histR[i] / totalPixels;
            entropyR -= pR * log2(pR);
        }
        if (histG[i] > 0) {
            double pG = (double)histG[i] / totalPixels;
            entropyG -= pG * log2(pG);
        }
        if (histB[i] > 0) {
            double pB = (double)histB[i] / totalPixels;
            entropyB -= pB * log2(pB);
        }
    }

    return (entropyR + entropyG + entropyB) / 3.0;
}

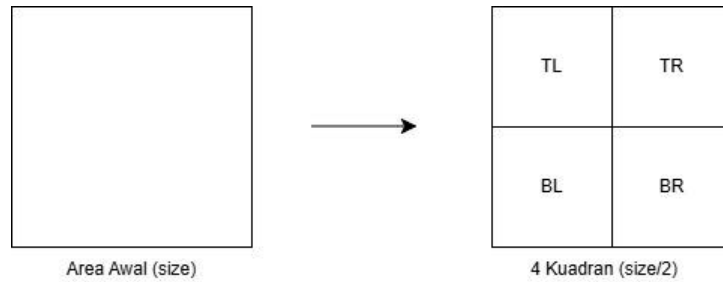
```

2.4 Algoritma *Divide and Conquer*

Algoritma divide and conquer dalam quadtree bekerja dengan prinsip:

1. **Divide**: Membagi masalah menjadi submasalah-submasalah yang lebih kecil (area gambar yang dibagi menjadi 4 kuadran)
2. **Conquer**: Menyelesaikan submasalah secara rekursif (memproses setiap kuadran)
3. **Combine**: Menggabungkan solusi dari submasalah (menghubungkan node-node menjadi struktur tree)

Visualisasi Pembagian Area:



Alur program:

```

QuadTreeNode* buildQuadTree(const vector<vector<Pixel>>& data, int x,
int y, int size, double threshold, int minBlockSize, int method, int
depth) {
    QuadTreeNode* node = new QuadTreeNode(x, y, size);

    Pixel avgColor = calculateAvgColor(data, x, y, size);
    node->avgColor = avgColor;

    double error =
calculateError(const_cast<vector<vector<Pixel>>&>(data), x, y, size,
avgColor, method);

    if (error > threshold && size > minBlockSize && size / 2 >=
minBlockSize) {
        node->isLeaf = false;
        int halfSize = size / 2;

        node->children[0] = buildQuadTree(data, x, y, halfSize,
threshold, minBlockSize, method, depth + 1);
        node->children[1] = buildQuadTree(data, x + halfSize, y,
halfSize, threshold, minBlockSize, method, depth + 1);
        node->children[2] = buildQuadTree(data, x, y + halfSize,
halfSize, threshold, minBlockSize, method, depth + 1);
        node->children[3] = buildQuadTree(data, x + halfSize, y +
halfSize, halfSize, threshold, minBlockSize, method, depth + 1);
    }

    return node;
}

```

2.5 Fungsi Tambahan

2.5.1 Konstruktor dan Destruktor

```

// QuadTreeNode constructor
QuadTreeNode::QuadTreeNode(int x, int y, int size) : x(x),
y(y), size(size), isLeaf(true) {
    for (int i = 0; i < 4; i++) {

```

```

        children[i] = nullptr;
    }
}

// QuadTreeNode destructor
QuadTreeNode::~QuadTreeNode() {
    for (int i = 0; i < 4; i++) {
        if (children[i]) {
            delete children[i];
        }
    }
}

```

2.5.2 Menghitung rata-rata warna dalam satu blok

```

Pixel calculateAvgColor(const vector<vector<Pixel>>& data,
int x, int y, int size) {
    long sumR = 0, sumG = 0, sumB = 0;
    int count = 0;

    for (int j = y; j < y + size && j < data.size(); j++) {
        for (int i = x; i < x + size && i < data[0].size();
i++) {
            sumR += data[j][i].r;
            sumG += data[j][i].g;
            sumB += data[j][i].b;
            count++;
        }
    }

    Pixel avg;
    if (count > 0) {
        avg.r = static_cast<unsigned char>(sumR / count);
        avg.g = static_cast<unsigned char>(sumG / count);
        avg.b = static_cast<unsigned char>(sumB / count);
    } else {
        avg.r = avg.g = avg.b = 0;
    }

    return avg;
}

```

2.5.3 Menghitung statistik blok untuk metode error

```

BlockStats calculateBlockStats(const vector<vector<Pixel>>&
data, int x, int y, int size) {

```

```

BlockStats stats;
long sumR = 0, sumG = 0, sumB = 0;
int count = 0;
unsigned char minR = 255, minG = 255, minB = 255;
unsigned char maxR = 0, maxG = 0, maxB = 0;

for (int j = y; j < y + size && j < data.size(); j++) {
    for (int i = x; i < x + size && i < data[0].size();
i++) {
        sumR += data[j][i].r;
        sumG += data[j][i].g;
        sumB += data[j][i].b;

        minR = min(minR, data[j][i].r);
        minG = min(minG, data[j][i].g);
        minB = min(minB, data[j][i].b);

        maxR = max(maxR, data[j][i].r);
        maxG = max(maxG, data[j][i].g);
        maxB = max(maxB, data[j][i].b);

        count++;
    }
}

if (count == 0) {
    return BlockStats{0};
}

stats.meanR = sumR / static_cast<double>(count);
stats.meanG = sumG / static_cast<double>(count);
stats.meanB = sumB / static_cast<double>(count);

stats.maxDiffR = maxR - minR;
stats.maxDiffG = maxG - minG;
stats.maxDiffB = maxB - minB;
double sumVarR = 0, sumVarG = 0, sumVarB = 0;
double sumMadR = 0, sumMadG = 0, sumMadB = 0;

map<unsigned char, int> histR, histG, histB;

for (int j = y; j < y + size && j < data.size(); j++) {
    for (int i = x; i < x + size && i < data[0].size();
i++) {
        double diffR = data[j][i].r - stats.meanR;
        double diffG = data[j][i].g - stats.meanG;

```

```

        double diffB = data[j][i].b - stats.meanB;
        sumVarR += diffR * diffR;
        sumVarG += diffG * diffG;
        sumVarB += diffB * diffB;
        sumMadR += abs(diffR);
        sumMadG += abs(diffG);
        sumMadB += abs(diffB);
        histR[data[j][i].r]++;
        histG[data[j][i].g]++;
        histB[data[j][i].b]++;
    }
}

stats.varianceR = sumVarR / count;
stats.varianceG = sumVarG / count;
stats.varianceB = sumVarB / count;

stats.madR = sumMadR / count;
stats.madG = sumMadG / count;
stats.madB = sumMadB / count;

// Calculate entropy
stats.entropyR = 0;
stats.entropyG = 0;
stats.entropyB = 0;

for (const auto& pair : histR) {
    double probability = pair.second /
static_cast<double>(count);
    stats.entropyR -= probability * log2(probability);
}

for (const auto& pair : histG) {
    double probability = pair.second /
static_cast<double>(count);
    stats.entropyG -= probability * log2(probability);
}

for (const auto& pair : histB) {
    double probability = pair.second /
static_cast<double>(count);
    stats.entropyB -= probability * log2(probability);
}

return stats;
}

```

2.5.4 Fungsi untuk menerima metode apa yang digunakan

```
double calculateError(vector<vector<Pixel>>& data, int x,
int y, int size, Pixel avgColor, int method) {
    switch (method) {
        case 1: // Variance
            return calculateVariance(data, x, y, size,
avgColor);
        case 2: // Mean Absolute Deviation (MAD)
            return calculateMAD(data, x, y, size, avgColor);
        case 3: // Max Pixel Difference
            return calculateMaxDifference(data, x, y, size);
        case 4: // Entropy
            return calculateEntropy(data, x, y, size);
        default:
            return calculateVariance(data, x, y, size,
avgColor); // Default to variance
    }
}
```

2.5.5 Fungsi rekonstruksi gambar

```
void reconstructImage(const QuadTreeNode* node,
vector<vector<Pixel>>& outputImage) {
    if (!node) return;
    if (node->isLeaf) {
        for (int j = node->y; j < node->y + node->size && j
< outputImage.size(); j++) {
            for (int i = node->x; i < node->x + node->size
&& i < outputImage[0].size(); i++) {
                outputImage[j][i] = node->avgColor;
            }
        }
    } else {
        for (int i = 0; i < 4; i++) {
            reconstructImage(node->children[i],
outputImage);
        }
    }
}
```

2.5.6 Fungsi menyimpan gambar

```
bool saveQuadTreeImage(const string& filename, const
vector<vector<Pixel>>& image) {
    int width = image[0].size();
    int height = image.size();
```

```

    unsigned char* buffer = new unsigned char[width * height
* 3];

    for (int y = 0; y < height; y++) {
        for (int x = 0; x < width; x++) {
            int idx = (y * width + x) * 3;
            buffer[idx] = image[y][x].r;
            buffer[idx + 1] = image[y][x].g;
            buffer[idx + 2] = image[y][x].b;
        }
    }

    string extension =
filename.substr(filename.find_last_of(".") + 1);
    transform(extension.begin(), extension.end(),
extension.begin(), ::tolower);

    bool success = false;
    if (extension == "png") {
        success = stbi_write_png(filename.c_str(), width,
height, 3, buffer, width * 3);
    } else if (extension == "jpg" || extension == "jpeg") {
        success = stbi_write_jpg(filename.c_str(), width,
height, 3, buffer, 90); // Quality 90
    } else if (extension == "bmp") {
        success = stbi_write_bmp(filename.c_str(), width,
height, 3, buffer);
    } else {
        cerr << "Unsupported output format: " << extension
<< endl;
    }

    delete[] buffer;
    return success;
}

```

2.5.7 Fungsi menghitung node dan kedalaman pohon

```

int countNodes(const QuadTreeNode* node) {
    if (!node) return 0;

    int count = 1; // Count this node

    if (!node->isLeaf) {
        for (int i = 0; i < 4; i++) {
            count += countNodes(node->children[i]);
        }
    }
}

```

```

        }
    }

    return count;
}

int getTreeDepth(const QuadTreeNode* node) {
    if (!node) return 0;

    if (node->isLeaf) {
        return 1;
    }

    int maxChildDepth = 0;
    for (int i = 0; i < 4; i++) {
        maxChildDepth = max(maxChildDepth,
getTreeDepth(node->children[i]));
    }

    return 1 + maxChildDepth;
}

```


BAB III

Uji Coba Program

3.1 Uji Coba

1. Variance

1.1 Threshold Variation

- Threshold: 50, 75, 100
- Minimum Block: 8x8 pixel (input: 8)
- Target Compression: Disabled (0.0)
- Output: branch11.jpg, branch12.jpg, branch13.jpg

1.2 Minimum Block Variation

- Threshold: 75
- Minimum Block: 4x4, 8x8, 16x16 pixel (input: 4, 8, 16)
- Target Compression: Disabled (0.0)
- Output: branch14.jpg, branch15.jpg, branch16.jpg

1.3 Target Compression Percentage Variation

- Initial Threshold: 75
- Minimum Block: 4x4 pixel (input: 4)
- Target Compression: 0.6, 0.8, 0.9
- Output: branch17.jpg, branch18.jpg, branch19.jpg

2. Mean Absolute Deviation (MAD)

2.1 Threshold Variation

- Threshold: 10, 15, 20
- Minimum Block: 4x4 pixel (input: 4)
- Target Compression: Disabled (0.0)
- Output: branch21.jpg, branch22.jpg, branch23.jpg

2.2 Minimum Block Variation

- Threshold: 8
- Minimum Block: 2x2, 4x4, 8x8 pixel (input: 2, 4, 8)
- Target Compression: Disabled (0.0)
- Output: branch24.jpg, branch25.jpg, branch26.jpg

2.3 Target Compression Percentage Variation

- Initial Threshold: 8
- Minimum Block: 4x4 pixel (input: 4)
- Target Compression: 0.6, 0.7, 0.86
- Output: branch27.jpg, branch28.jpg, branch29.jpg

3. Max Pixel Difference

3.1 Threshold Variation

- Threshold: 30, 40, 50
- Minimum Block: 4x4 pixel (input: 4)
- Target Compression: Disabled (0.0)
- Output: branch31.jpg, branch32.jpg, branch33.jpg

3.2 Minimum Block Variation

- Threshold: 40
- Minimum Block: 4x4, 8x8, 16x16 pixel (input: 4, 8, 16)
- Target Compression: Disabled (0.0)

- Output: branch34.jpg, branch35.jpg, branch36.jpg

3.3 Target Compression Percentage Variation

- Initial Threshold: 40
- Minimum Block: 4x4 pixel (input: 4)
- Target Compression: 0.6, 0.75, 0.9
- Output: branch37.jpg, branch38.jpg, branch39.jpg

4. Entropy

4.1 Threshold Variation

- Threshold: 1.5, 5, 9
- Minimum Block: 16x16 pixel (input: 16)
- Target Compression: Disabled (0.0)
- Output: branch41.jpg, branch42.jpg, branch43.jpg

4.2 Minimum Block Variation

- Threshold: 3
- Minimum Block: 6x6, 8x8, 10x10 pixel (input: 6, 8, 10)
- Target Compression: Disabled (0.0)
- Output: branch44.jpg, branch45.jpg, branch46.jpg

4.3 Target Compression Percentage Variation

- Initial Threshold: 4.5
- Minimum Block: 8x8 pixel (input: 8)
- Target Compression: 0.75, 0.85, 0.95
- Output: branch47.jpg, branch48.jpg, branch49.jpg

5. Uji Coba Khusus Gambar Kontras untuk metode Max Pixel Difference dan Variance

5.1. Max Pixel Difference

- Threshold: 100
 - Minimum Block: 16x16 pixel (input: 64)
 - Target Compression: Disabled (0.0)
 - Output: blackwhite1.jpg
- Execution time: 21.88 seconds
 Original image size: 11059200 bytes
 Compressed size (approximate): 316080 bytes
 Compression percentage: 97.14%
 Tree depth: 8
 Number of nodes: 6585

5.2. Variance

- Threshold: 100
 - Minimum Block: 16x16 pixel (input: 64)
 - Target Compression: Disabled (0.0)
 - Output: blackwhite2.jpg
- Execution time: 18.60 seconds
 Original image size: 11059200 bytes
 Compressed size (approximate): 313200 bytes
 Compression percentage: 97.17%
 Tree depth: 8
 Number of nodes: 652

3.2 Hasil Uji Coba

Keterangan: Yang ditampilkan pada laporan hanya dua per metode error, selebihnya dapat dilihat pada repo github pada folder test.

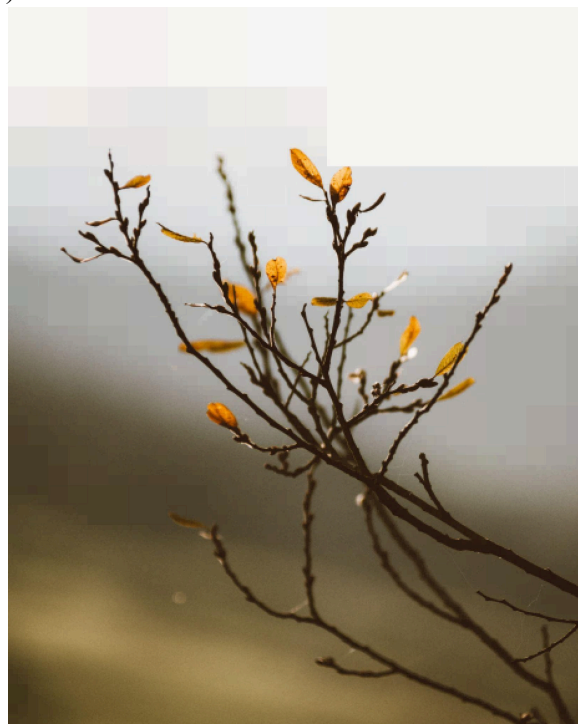
Detail Spesifikasi Hasil

branch.jpg (gambar awal)



branch.jpg

branch11.jpg (variance)



branch11.jpg

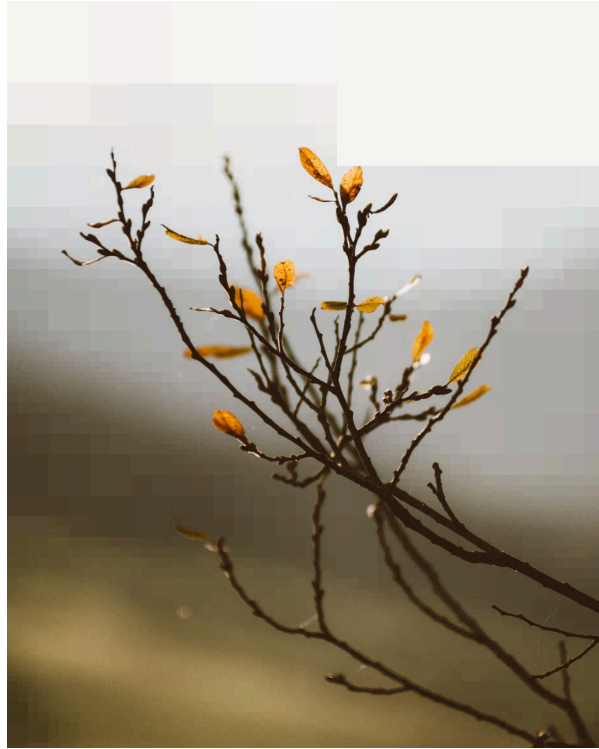
Execution time: 28.22 seconds

Original image size: 51671040 bytes

Compressed size (approximate): 6348720 bytes

Compression percentage: 87.71%
Tree depth: 11
Number of nodes: 132265

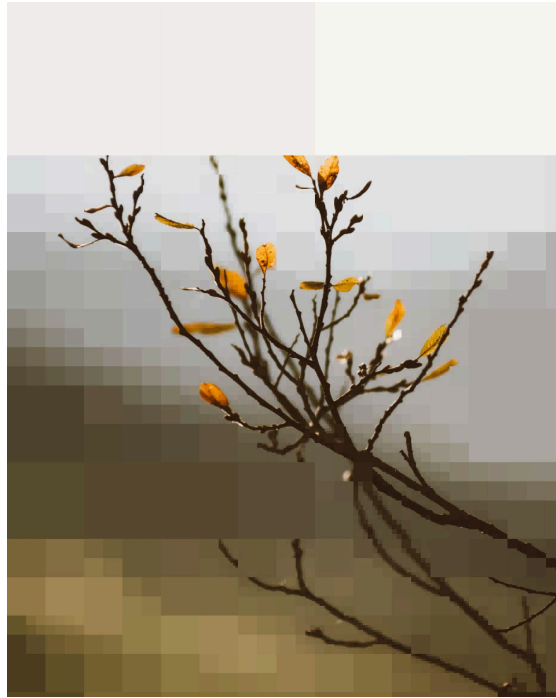
branch19.jpg (variance + bonus target kompresi : 80%)



branch19.jpg

Target compression reached with threshold: 57.5605
Execution time: 43.71 seconds
Original image size: 51671040 bytes
Compressed size (approximate): 9853104 bytes
Compression percentage: 80.93%
Tree depth: 12
Number of nodes: 205273

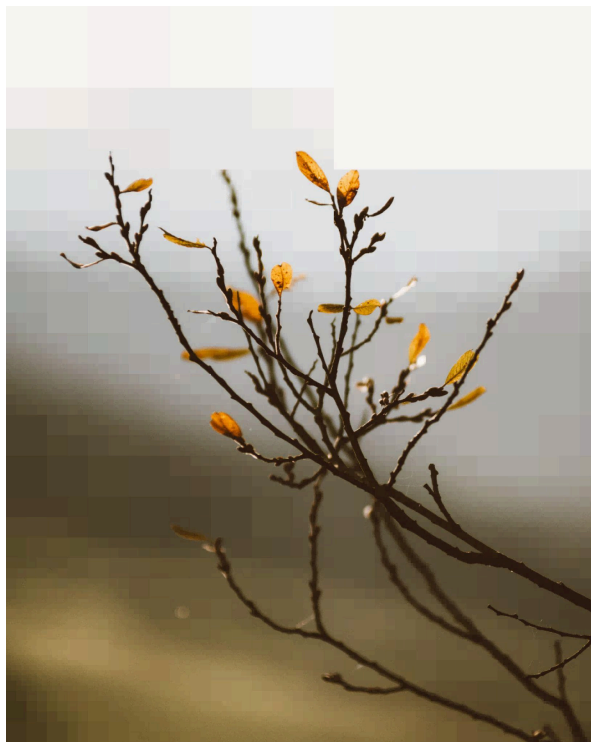
branch21.jpg (MAD)



branch21.jpg

Execution time: 27.70 seconds
Original image size: 51671040 bytes
Compressed size (approximate): 2975856 bytes
Compression percentage: 94.24%
Tree depth: 12
Number of nodes: 61997

branch29.jpg (MAD + bonus target kompresi : 86%)



branch29.jpg

Target compression reached with threshold: 6.2856
Execution time: 61.90 seconds

Original image size: 51671040 bytes
Compressed size (approximate): 6869808 bytes
Compression percentage: 86.70%
Tree depth: 12
Number of nodes: 143121

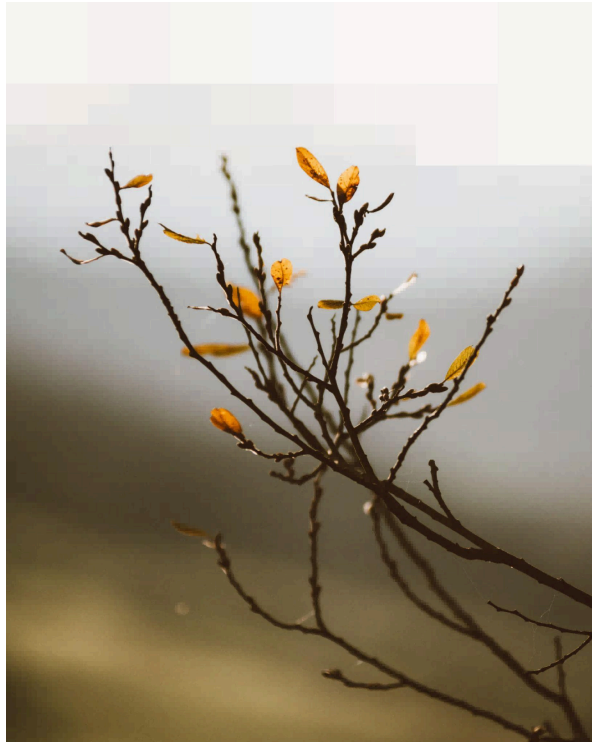
branch31.jpg (max pixel difference)



branch31.jpg

Execution time: 56.28 seconds
Original image size: 51671040 bytes
Compressed size (approximate): 33690864 bytes
Compression percentage: 34.80%
Tree depth: 12
Number of nodes: 701893

branch39.jpg (max pixel difference + bonus target kompresi : 90%)



branch39.jpg

Target compression reached with threshold: 49.1907

Execution time: 85.31 seconds

Original image size: 51671040 bytes

Compressed size (approximate): 5502384 bytes

Compression percentage: 89.35%

Tree depth: 12

Number of nodes: 114633

branch41.jpg (Entropy)

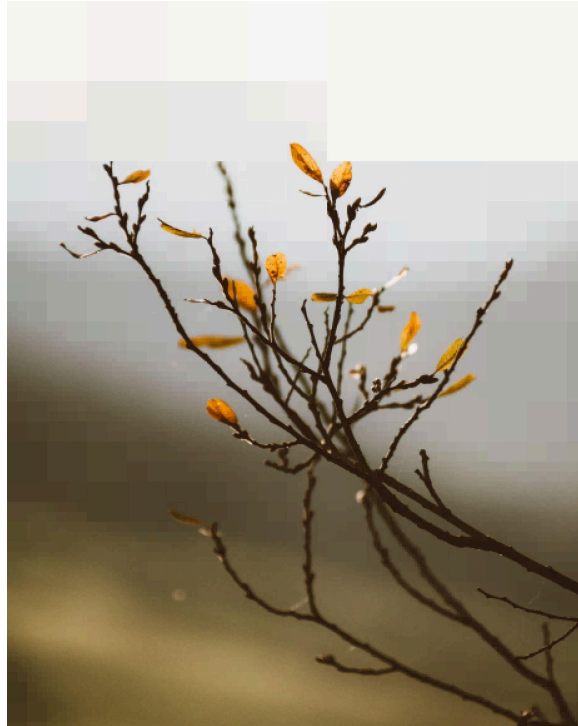


branch41.jpg

Execution time: 18.63 seconds

Original image size: 51671040 bytes
Compressed size (approximate): 4324080 bytes
Compression percentage: 91.63%
Tree depth: 10
Number of nodes: 90085

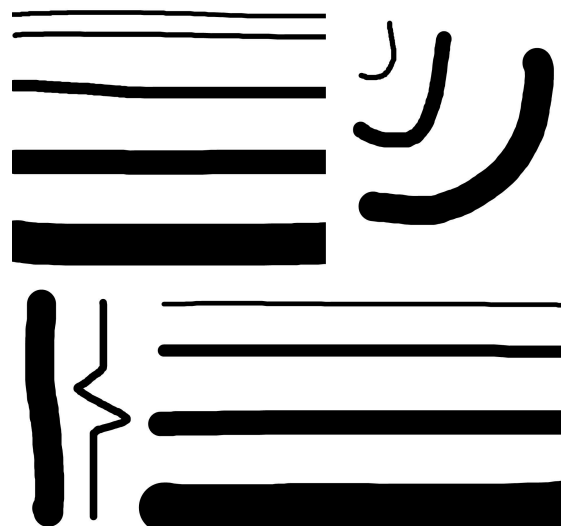
branch49.jpg (Entropi + bonus target kompresi : 95%)



branch49.jpg

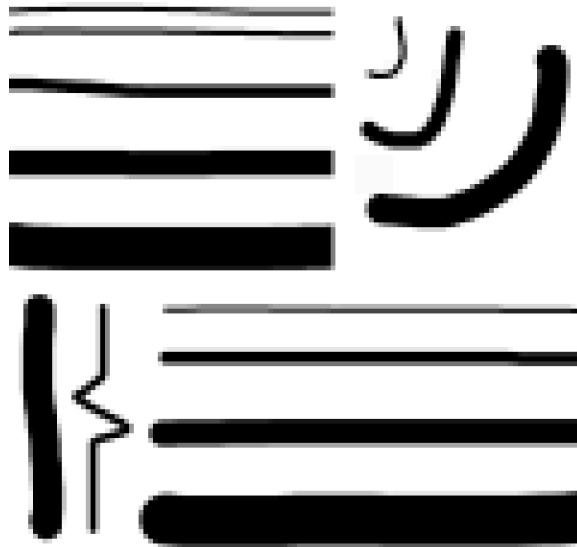
Target compression reached with threshold: 4.92608
Execution time: 30.62 seconds
Original image size: 51671040 bytes
Compressed size (approximate): 3057840 bytes
Compression percentage: 94.08%
Tree depth: 11
Number of nodes: 63705

blackwhite.jpg (gambar awal)



blackwhite.jpg

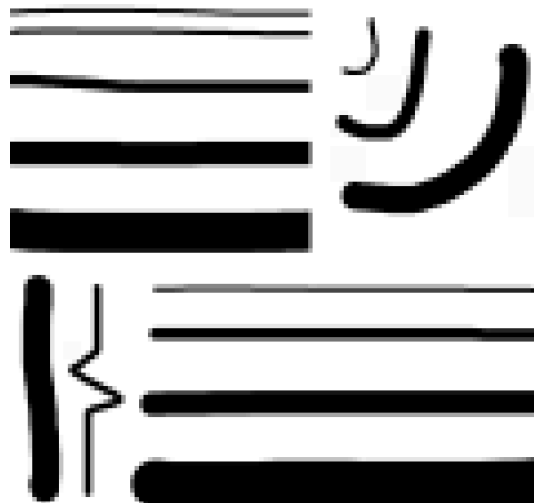
blackwhite1.jpg (max pixel difference)



blackwhite1.jpg

Execution time: 21.88 seconds
Original image size: 11059200 bytes
Compressed size (approximate): 316080 bytes
Compression percentage: 97.14%
Tree depth: 8
Number of nodes: 6585

blackwhite2.jpg (variance)



blackwhite2.jpg

Execution time: 18.60 seconds
Original image size: 11059200 bytes
Compressed size (approximate): 313200 bytes
Compression percentage: 97.17%
Tree depth: 8
Number of nodes: 652

1. Uji Coba Metode Error

1) Variance

Variance cocok untuk gambar dengan transisi halus. Ambang tinggi tingkatan kompresi tapi bisa hilangkan detail. Blok kecil pertahankan detail tapi kurang efisien. Kompresi target fleksibel tapi lambat.

2) Mean Absolute Derivation

MAD tahan terhadap outlier, cocok untuk gambar dengan variasi sedang. Ambang tinggi dan blok besar tingkatan efisiensi, tapi kompresi target lambat.

3) Max Pixel Difference

Sensitif terhadap perbedaan ekstrem, cocok untuk gambar dengan batas jelas. Ambang rendah kurang efisien, blok besar lebih baik, dan kompresi target lambat.

4) Entrophy

Cocok untuk gambar kompleks. Ambang tinggi bisa hilangkan detail, blok besar kurang berpengaruh, dan kompresi target efisien.

2. Uji Coba Gambar Kontras pada metode Max Pixel Difference

Metode Max Pixel Difference mengukur perbedaan intensitas terbesar dalam blok. Pada gambar kontras tinggi (misalnya hitam-putih), perbedaan besar (misalnya 255) memicu pembagian blok jika melebihi ambang (100), menghasilkan lebih banyak node (6.585) dibandingkan Variance (652). Variance lebih efisien karena rata-rata intensitas menghasilkan blok homogen lebih cepat. Max Pixel Difference lebih baik untuk menjaga detail tepi, tapi kurang efisien untuk kompresi tinggi.

Gambar kontras memiliki area seragam besar (hitam/putih), sehingga quadtree membentuk blok besar dengan sedikit pembagian. Variance unggul dalam efisiensi, sedangkan Max Pixel Difference lebih detail pada tepi.

BAB IV

Kesimpulan dan Saran

4.1 Kesimpulan

Algoritma *divide and conquer* dalam kompresi gambar berbasis quadtree efektif karena memecah gambar menjadi blok-blok kecil secara rekursif, lalu mengevaluasi homogenitasnya berdasarkan metrik error seperti Variance, MAD, Max Pixel Difference, atau Entropy. Pendekatan ini memungkinkan kompresi tinggi dengan menggabungkan blok seragam menjadi representasi sederhana, terutama pada gambar kontras tinggi, sambil tetap mempertahankan detail pada area kompleks melalui pembagian lebih lanjut, meskipun dengan biaya komputasi yang lebih tinggi.

Namun, efisiensi algoritma bergantung pada pengaturan parameter seperti ambang batas dan ukuran blok minimum. Ambang tinggi dan blok besar meningkatkan kompresi tetapi dapat mengorbankan detail, sedangkan kompresi target menawarkan fleksibilitas dengan waktu eksekusi lebih lama. Secara keseluruhan, *divide and conquer* memberikan keseimbangan antara kualitas dan kompresi, dengan Variance dan MAD cocok untuk gambar umum, Max Pixel Difference untuk kontras tinggi, dan Entropy untuk tekstur kompleks.

4.2 Saran

1. Sebaiknya disertakan beberapa file .jpg sebagai testcase, agar kami memiliki gambaran terkait standar kualitas keluaran program.
2. Program ini dapat dikembangkan kedepannya agar bisa lebih baik dan dapat digunakan untuk kebutuhan masa depan.

Referensi

York, T. (2020). *Quadrees for image processing*. Medium.

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

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 1)*. Diakses pada 9 April 2024, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-\(2025\)-Bagian1.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/07-Algoritma-Divide-and-Conquer-(2025)-Bagian1.pdf)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 2)*. Diakses pada 9 April 2024, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-\(2025\)-Bagian2.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/08-Algoritma-Divide-and-Conquer-(2025)-Bagian2.pdf)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 3)*. Diakses pada 9 April 2024, dari

[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-\(2025\)-Bagian3.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/09-Algoritma-Divide-and-Conquer-(2025)-Bagian3.pdf)

Munir, R. (2024). *Algoritma Divide and Conquer (Bagian 4)*. Diakses pada 9 April 2024, dari


[https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-\(2025\)-Bagian4.pdf](https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2024-2025/10-Algoritma-Divide-and-Conquer-(2025)-Bagian4.pdf)

Lampiran

Link menuju *repository* Github:

[ivant8k/Tucil2_13523127_13523129](https://github.com/ivant8k/Tucil2_13523127_13523129)

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

8. Program dan laporan dibuat (kelompok) sendiri		
--	---	--