

Laporan Tugas Kecil 2 IF2211 Strategi Algoritma

Sem II tahun 2024/2025

Kompresi Gambar Dengan Metode Quadtree



Disusun oleh :

Fityatul Haq Rosyidi - 13523116

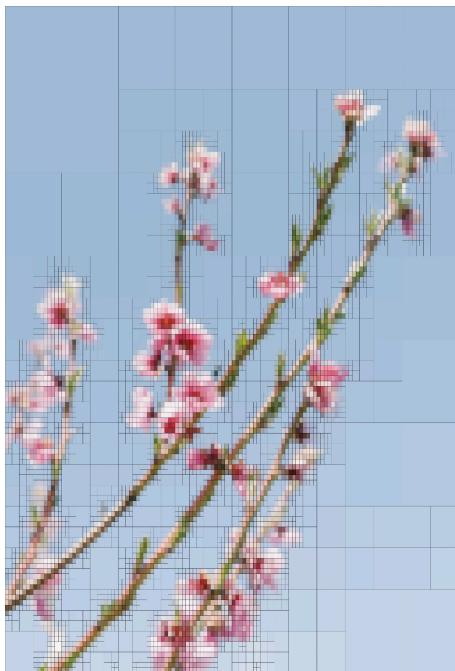
**Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung**

2025

Daftar Isi

Daftar Isi.....	2
BAB I : Deskripsi Tugas.....	3
BAB II : Rancangan Algoritma.....	7
BAB III : Implementasi.....	8
Paradigma.....	8
Struktur Data.....	8
Struktur Repository.....	8
Snapshot Code.....	9
BAB IV : Analisis Kompleksitas.....	18
BAB V : Testing.....	19
BAB VI : Lampiran.....	21
Referensi.....	22

BAB I : Deskripsi Tugas



Gambar 2. Quadtree dalam Kompresi Gambar

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.

Ide pada tugas kecil 2 ini cukup sederhana, seperti pada pembahasan sebelumnya mengenai Quadtree. Berikut adalah prosedur pada program kompresi gambar yang akan dibuat dalam Tugas Kecil 2 (Divide and Conquer):

1. Inisialisasi dan Persiapan Data

Masukkan gambar yang akan dikompresi akan diolah dalam format matriks piksel dengan nilai intensitas berdasarkan sistem warna RGB. Berikut adalah parameter-parameter yang dapat ditentukan oleh pengguna saat ingin melakukan kompresi gambar:

- a. Metode perhitungan variansi
- b. Threshold variansi: nilai ambang batas untuk menentukan apakah blok akan dibagi lagi.
- c. Minimum block size: ukuran minimum blok piksel yang diperbolehkan untuk diproses lebih lanjut.

2. Perhitungan Error

Untuk setiap blok gambar yang sedang diproses, hitung nilai variansi menggunakan metode yang dipilih sesuai Tabel 1.

3. Pembagian Blok

Bandingkan nilai variansi blok dengan threshold:

- Jika variansi di atas threshold (cek kasus khusus untuk metode bonus), ukuran blok lebih besar dari minimum block size, dan ukuran blok setelah dibagi menjadi empat kurang dari minimum block size, blok tersebut dibagi menjadi empat sub-blok, dan proses dilanjutkan untuk setiap sub-blok.
- Jika salah satu kondisi di atas tidak terpenuhi, proses pembagian dihentikan untuk blok tersebut.

4. Normalisasi Warna

Untuk blok yang tidak lagi dibagi, lakukanlah normalisasi warna blok sesuai dengan rata-rata nilai RGB blok.

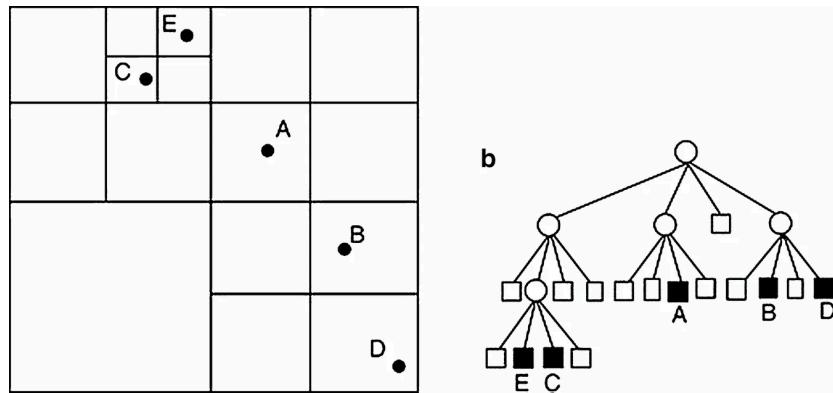
5. Rekursi dan Penghentian

Proses pembagian blok dilakukan secara rekursif untuk setiap sub-blok hingga semua blok memenuhi salah satu dari dua kondisi berikut:

- Error blok berada di bawah threshold.
- Ukuran blok setelah dibagi menjadi empat kurang dari minimum block size.

6. Penyimpanan dan Output

Rekonstruksi gambar dilakukan berdasarkan struktur QuadTree yang telah dihasilkan selama proses kompresi. Gambar hasil rekonstruksi akan disimpan sebagai file terkompresi. Selain itu, persentase kompresi akan dihitung dan disertakan dengan rumus sesuai dengan yang terlampir pada dokumen ini. Persentase kompresi ini memberikan gambaran mengenai efisiensi metode kompresi yang digunakan.



Gambar 2. Struktur Data Quadtree dalam Kompresi Gambar

BAB II : Rancangan Algoritma

Algoritma yang digunakan pada tugas kecil kali ini adalah algoritma divide and conquer juga struktur data Quadtree, yakni dengan membagi blok gambar menjadi blok-blok yang lebih kecil seterusnya sehingga blok mencapai batas tertentu. Berikut adalah langkah-langkah lebih detail terkait algoritma divide and conquer yang digunakan :

1. User memasukkan input berupa
 - a. Alamat absolut gambar yang akan dikompresi
 - b. Metode perhitungan Error
 - c. Threshold
 - d. Ukuran block minimum
 - e. Alamat absolut gambar hasil kompresi
2. Dilakukan pengukuran warna rata-rata pada blok gambar saat ini
3. Dilakukan Pengukuran Error (Error Measurement) berdasarkan metode yang dipilih. Terdapat 4 metode yang tersedia
 - a. Variance
 - b. Mean Absolute Value (MAD)
 - c. Max Pixel Difference
 - d. EntropyHasil pengukuran disimpan ke variabel bernama **variance**
4. Diperiksa apakah salah satu dari dua hal ini terpenuhi
 - a. Ukuran blok kurang dari Ukuran Blok Minimum
 - b. Nilai **variance** kurang dari thresholdJika ada minimal satu saja yang terpenuhi, “warnai” seluruh piksel pada blok gambar output di posisi serupa dengan warna rata-rata yang diperoleh sebelumnya. Algoritma (pada cabang ini) selesai. Jika tidak, lanjut ke langkah ke-5
5. Bagi blok gambar menjadi 4 kuadran/bagian yang sama besar, kemudian ulangi algoritma dari langkah ke-2 pada masing-masing kuadran/bagian.

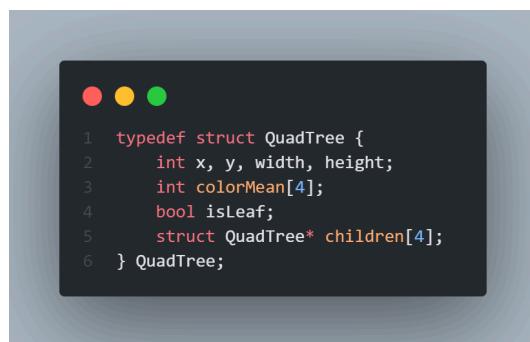
BAB III : Implementasi

Paradigma

Implementasi dilakukan dengan paradigma **Prosedural** menggunakan bahasa **C**. penulis memilih menggunakan paradigma prosedural karena program yang diminta tidak terlalu kompleks, ditambah penulis lebih familiar dengan paradigma ini daripada paradigma-paradigma pemrograman yang lain sehingga akan lebih mudah untuk menulis program.

Struktur Data

Seperti yang diminta pada spesifikasi tugas kecil ini, struktur data yang digunakan untuk memecah citra adalah Quadtree. Quadtree adalah pohon dengan maksimal 4 anak. Quadtree pada program ini dibuat sebagai **ADT** yang memiliki atribut posisi (x, y), ukuran (panjang, lebar), rata-rata warna dalam bentuk list statik (R, G, B, A), indikator daun, dan pointer ke Quadtree anak. Lebih lengkap dapat dilihat pada snapshot code berikut



```
1  typedef struct QuadTree {
2      int x, y, width, height;
3      int colorMean[4];
4      bool isLeaf;
5      struct QuadTree* children[4];
6  } QuadTree;
```

Struktur Repository

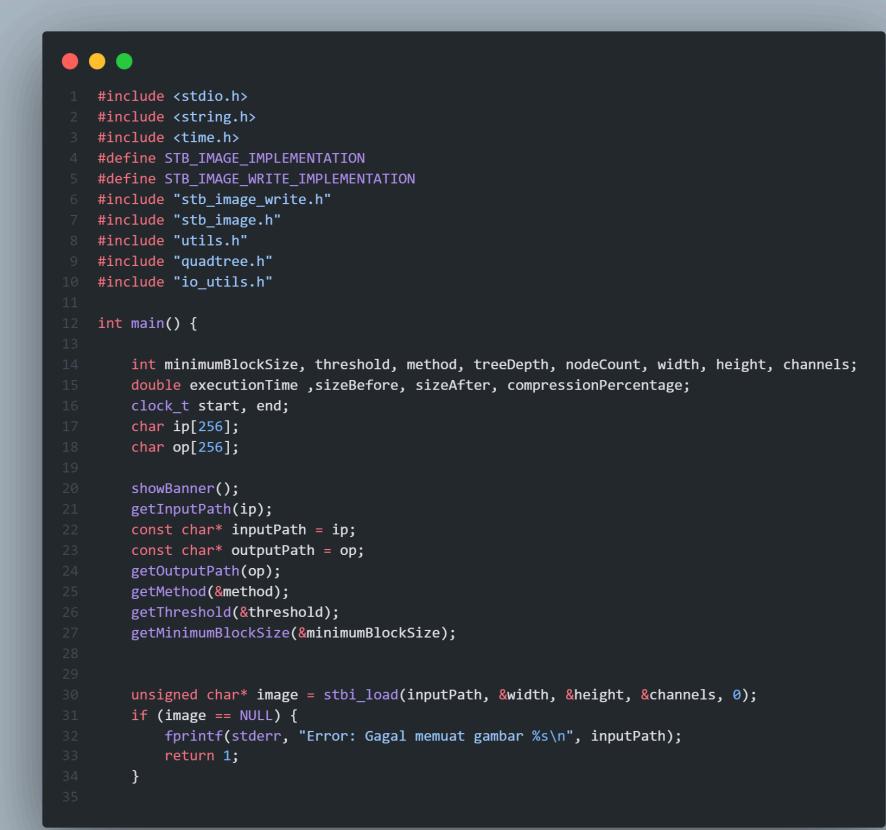
Repository berisi bin, doc, src, dan test. Yang akan lebih dibahas adalah file-file dalam folder src. Folder src terdiri dari

- main.c : program utama
- utils.c : library utama yang berisi fungsi-fungsi matematis dan fungsi rekursif utama (DAC)
- Io_utils.c : library untuk fungsi-fungsi input dan output
- quadtree .c : library/file implementasi ADT quadtree

Selain itu, terdapat juga beberapa file header seperti

- stb_image.h : library eksternal untuk manipulasi file citra
- stb_image_write.h : library eksternal untuk menulis file citra
- utils.h : header file utils.c
- io_utils.h : header file io_utils.h
- quadtree.h : header file quadtree.h

Snapshot Code

Nama File	Snapshot
main.c	 <pre>● ● ● 1 #include <stdio.h> 2 #include <string.h> 3 #include <time.h> 4 #define STB_IMAGE_IMPLEMENTATION 5 #define STB_IMAGE_WRITE_IMPLEMENTATION 6 #include "stb_image_write.h" 7 #include "stb_image.h" 8 #include "utils.h" 9 #include "quadtree.h" 10 #include "io_utils.h" 11 12 int main() { 13 14 int minimumBlockSize, threshold, method, treeDepth, nodeCount, width, height, channels; 15 double executionTime ,sizeBefore, sizeAfter, compressionPercentage; 16 clock_t start, end; 17 char ip[256]; 18 char op[256]; 19 20 showBanner(); 21 getInputPath(ip); 22 const char* inputPath = ip; 23 const char* outputPath = op; 24 getOutputPath(op); 25 getMethod(&method); 26 getThreshold(&threshold); 27 getMinimumBlockSize(&minimumBlockSize); 28 29 30 unsigned char* image = stbi_load(inputPath, &width, &height, &channels, 0); 31 if (image == NULL) { 32 fprintf(stderr, "Error: Gagal memuat gambar %s\n", inputPath); 33 return 1; 34 } 35 }</pre>

```
1 unsigned char* newImage = (unsigned char*)malloc(channels * width * height);
2 if (image == NULL) {
3     fprintf(stderr, "Error: Gagal mengalokasikan memori untuk gambar.\n");
4     exit(1);
5 }
6
7 // Root Tree
8 QuadTree* root = createNode(0, 0, width, height, image, channels, width);
9
10 // Divide and Conquer Algorithm
11 start = clock();
12 DAC(minimumBlockSize, root, image, newImage, channels, threshold, width, method);
13 end = clock();
14
15
16 // tulis gambar ke file
17 int success = stbi_write_jpg(outputPath, width, height, channels, newImage, 50);
18
19 // outputs
20 executionTime = ((double)(end - start)) / CLOCKS_PER_SEC;
21 sizeBefore = (double) getFileSize(inputPath) / 1024.0;
22 sizeAfter = (double) getFileSize(outputPath) / 1024.0;
23 compressionPercentage = (1 - (sizeAfter / sizeBefore)) * 100;
24 treeDepth = getTreeDepth(root);
25 nodeCount = getNodeCount(root);
26
27 printf("Waktu Eksekusi : %.2f sec\n", executionTime);
28 printf("size before : %.2f KB\n", sizeBefore);
29 printf("size after : %.2f KB\n", sizeAfter);
30 printf("compression percentage : %.2f%% \n", compressionPercentage);
31 printf("Tree Depth : %d\n", treeDepth);
32 printf("Tree node count : %d\n", nodeCount);
33
```

```
1     printf("Waktu Eksekusi : %.2f sec\n", executionTime);
2     printf("size before : %.2f KB\n", sizeBefore);
3     printf("size after : %.2f KB\n", sizeAfter);
4     printf("compression percentage : %.2f%% \n", compressionPercentage);
5     printf("Tree Depth : %d\n", treeDepth);
6     printf("Tree node count : %d\n", nodeCount);
7
8     if (success) {
9         printf("Gambar berhasil disimpan di %s\n", outputPath);
10    } else {
11        fprintf(stderr, "Error: Gagal menyimpan gambar.\n");
12    }
13
14 // Bebaskan memori setelah selesai
15 stbi_image_free(image);
16 free(newImage);
17 freeQuadTree(root);
18
19 return 0;
20 }
```

io_utils.c

```
1 #include "io_utils.h"
2
3 void showBanner() {
4     printf("=====\\n");
5     printf("| TUCIL 2 STIMA |\\n");
6     printf("| QUADTREE COMPRESION |\\n");
7     printf("=====\\n");
8     printf(" by : yayatsigma \\n");
9     printf("\\n\\n");
10 }
11
12 void getInputPath(char* alamat) {
13     // Meminta get untuk alamat dengan validasi
14     while (1) {
15         printf("Masukkan alamat absolut file input \\n> ");
16         if (fgets(alamat, 256, stdin) != NULL) {
17             // Menghapus newline jika ada
18             alamat[strcspn(alamat, "\\n")] = '\\0';
19             if (strlen(alamat) > 0) {
20                 break; // Jika alamat tidak kosong, keluar dari loop
21             } else {
22                 printf("Error: Alamat tidak boleh kosong.\\n");
23             }
24         } else {
25             printf("Error: Gagal membaca alamat.\\n");
26         }
27     }
28 }
```

```
1 void getOutputPath(char* alamat) {
2     // Meminta get untuk alamat dengan validasi
3     while (1) {
4         printf("Masukkan alamat absolut file output \n>> ");
5         if (fgets(alamat, 256, stdin) != NULL) {
6             // Menghapus newline jika ada
7             alamat[strcspn(alamat, "\n")] = '\0';
8             if (strlen(alamat) > 0) {
9                 break; // Jika alamat tidak kosong, keluar dari loop
10            } else {
11                printf("Error: Alamat tidak boleh kosong.\n");
12            }
13        } else {
14            printf("Error: Gagal membaca alamat.\n");
15        }
16    }
17 }
18
19
20 void getMethod(int* metode) {
21     int status;
22     printf("=====\\n");
23     printf("Error Measurement Methods:\\n");
24     printf("1. Variance\\n");
25     printf("2. Mean Absolute Deviation (MAD)\\n");
26     printf("3. Max Pixel Different\\n");
27     printf("4. Entropy\\n");
28     printf("=====\\n");
29     while (1) {
30         printf("Masukkan metode (1 - 4) \\n>> ");
31         status = scanf("%d", metode);
32         if (status == 1 && *metode > 0 && *metode < 5) {
33             break; // Jika get valid, keluar dari loop
34         } else {
35             printf("Error: metode harus berupa integer antara 1 - 4.\n");
36             while (getchar() != '\n'); // Membersihkan buffer get
37         }
38     }
39 }
40 }
```

```

1 void getThreshold(int* threshold) {
2     int status;
3     while (1) {
4         printf("Masukkan threshold (integer) \n>> ");
5         status = scanf("%d", threshold);
6         if (status == 1) {
7             break; // Jika get valid, keluar dari loop
8         } else {
9             printf("Error: threshold harus berupa integer.\n");
10            while (getchar() != '\n'); // Membersihkan buffer get
11        }
12    }
13 }
14
15
16 void getMinimumBlockSize(int* minimumBlockSize) {
17     int status;
18     while (1) {
19         printf("Masukkan Ukuran blok minimum (integer) \n>> ");
20         status = scanf("%d", minimumBlockSize);
21         if (status == 1) {
22             break; // Jika get valid, keluar dari loop
23         } else {
24             printf("Error: get harus berupa integer.\n");
25             while (getchar() != '\n'); // Membersihkan buffer get
26         }
27     }
28 }
```

quadtree.c

```

1 #include "quadtree.h"
2 #include "utils.h"
3
4 QuadTree* createNode(int x, int y, int width, int height, unsigned char* image, int channels, int absWidth) {
5     QuadTree* newNode = (QuadTree*)malloc(sizeof(QuadTree));
6     if (newNode == NULL) {
7         fprintf(stderr, "Error: Gagal mengalokasikan memori");
8         exit(1);
9     }
10
11    newNode->x = x;
12    newNode->y = y;
13    newNode->wWidth = width;
14    newNode->height = height;
15    newNode->isLeaf = true;
16
17    // inisiasi array color mean
18    newNode->colorMean[0] = calculate_mean(image, x, y, width, height, channels, 0, absWidth);
19    newNode->colorMean[1] = calculate_mean(image, x, y, width, height, channels, 1, absWidth);
20    newNode->colorMean[2] = calculate_mean(image, x, y, width, height, channels, 2, absWidth);
21
22
23    // Inisialisasi children ke NULL
24    for (int i = 0; i < 4; i++) {
25        newNode->children[i] = NULL;
26    }
27
28    return newNode;
29 }
```

```
1 void connect(QuadTree* parent, QuadTree* child, int index) {
2     if (parent == NULL) {
3         fprintf(stderr, "Error: Node induk tidak boleh NULL.\n");
4         return;
5     }
6     if (index < 0 || index >= 4) {
7         fprintf(stderr, "Error: Indeks anak harus antara 0 dan 3.\n");
8         return;
9     }
10    parent->children[index] = child;
11    parent->isLeaf = false;
12}
13
14
15 void freeQuadTree(QuadTree* root) {
16     if (root == NULL) {
17         return;
18     }
19
20     for (int i = 0; i < 4; i++) {
21         freeQuadTree(root->children[i]);
22     }
23}
24
```

```
1
2 int getTreeDepth(QuadTree* root) {
3     if (root == NULL) {
4         return 0; // Kedalaman dari tree kosong adalah 0
5     }
6
7     int maxDepth = 0;
8     for (int i = 0; i < 4; i++) {
9         int childDepth = getTreeDepth(root->children[i]);
10        if (childDepth > maxDepth) {
11            maxDepth = childDepth;
12        }
13    }
14
15    return 1 + maxDepth; // Menambahkan 1 untuk menghitung level saat ini
16 }
17
18 int getNodeCount(QuadTree* root) {
19     if (root == NULL) {
20         return 0; // Jika node kosong, jumlah simpul adalah 0
21     }
22
23     int count = 1; // Hitung node saat ini
24     for (int i = 0; i < 4; i++) {
25         count += getNodeCount(root->children[i]); // Tambahkan jumlah node dari subtree
26     }
27
28     return count;
29 }
```

utils.c

```
● ● ●
1 #include "utils.h"
2 #include "stb_image.h"
3
4
5 int calculate_mean(unsigned char* image, int x, int y, int width, int height, int channels, int idx, int absWidth) {
6     assert(image != NULL);
7
8     long long sum = 0;
9     int pixelCount = 0;
10
11    for (int j = 0; j < height ; ++j) {
12        for (int i = 0; i < width; ++i) {
13            int index = ((y + j) * absWidth + (x + i)) * channels;
14            sum = sum + image[index + idx];
15            pixelCount++;
16        }
17    }
18
19    return (sum / pixelCount);
20}
21
22 double calculate_variance(unsigned char* image, int x, int y, int width, int height, int channels, int* colorMean, int absWidth) {
23     //jumlah variansi semua channel
24     int sum = 0;
25
26     //jumlah variansi per satuan channel
27     int oneChannelSum = 0;
28
29     //banyak pixel dalam satu blok
30     int N = width * height;
31
32     for (int c = 0; c < channels; c++) {
33         // hitung variansi masing2 channel
34
35         for (int j = 0; j < height ; ++j) {
36             for (int i = 0; i < width; ++i) {
37                 int index = ((y + j) * absWidth + (x + i)) * channels;
38                 int colorValue = (int)image[index + c];
39                 oneChannelSum += pow(colorValue - colorMean[c], 2);
40             }
41         }
42     }
43 }
```

```
● ● ●
1 double calculate_MAD(unsigned char* image, int x, int y, int width, int height, int channels, int* colorMean, int absWidth) {
2     //jumlah variansi semua channel
3     int sum = 0;
4
5     //jumlah variansi per satuan channel
6     int oneChannelSum = 0;
7
8     //banyak pixel dalam satu blok
9     int N = width * height;
10
11    for (int c = 0; c < channels; c++) {
12        // hitung variansi masing2 channel
13
14        for (int j = 0; j < height ; ++j) {
15            for (int i = 0; i < width; ++i) {
16                int index = ((y + j) * absWidth + (x + i)) * channels;
17                int colorValue = (int)image[index + c];
18                oneChannelSum += abs(colorValue - colorMean[c]);
19            }
20        }
21        // disini variansi per warna sudah ketemu
22        sum += (oneChannelSum / N);
23        // reset
24        oneChannelSum = 0;
25    }
26    return (double)sum / 3.0;
27 }
```

```

1 double calculate_MPDI(unsigned char* image, int x, int y, int width, int height, int channels, int* colorMean, int absWidth) {
2     int difference = 0;
3     int oneChannelDifference = 0;
4     int maxPixel, minPixel;
5
6     for (int c = 0; c < channels; c++) {
7         maxPixel = (int)image[(y * absWidth + x) * channels + c];
8         minPixel = (int)image[(y * absWidth + x) * channels + c];
9         for (int j = 0; j < height; ++j) {
10             for (int i = 0; i < width; ++i) {
11                 int index = ((y + j) * absWidth + (x + i)) * channels;
12                 int colorValue = (int)image[index + c];
13
14                 if (colorValue > maxPixel) {
15                     maxPixel = colorValue;
16                 }
17                 if (colorValue < minPixel) {
18                     minPixel = colorValue;
19                 }
20             }
21         }
22         oneChannelDifference = maxPixel - minPixel;
23         // disini variansi per warna sudah ketemu
24         difference += oneChannelDifference;
25         // reset
26         oneChannelDifference = 0;
27     }
28     return (double)difference / 3.0;
29 }
```

```

1 double calculate_Entropy(unsigned char* image, int x, int y, int width, int height, int channels, int* colorMean, int absWidth) {
2
3     double entropyR, entropyG, entropyB, entropyTotal;
4     int frequencyTableR[256] = {0};
5     int frequencyTableG[256] = {0};
6     int frequencyTableB[256] = {0};
7     double probabilityR, probabilityG, probabilityB;
8     int count, N;
9
10    N = width * height;
11
12    // looping pertama untuk mengisi tabel frekuensi
13    for (int j = 0; j < height; ++j) {
14        for (int i = 0; i < width; ++i) {
15            int index = ((y + j) * absWidth + (x + i)) * channels;
16            int colorValueR = (int)image[index];
17            int colorValueG = (int)image[index + 1];
18            int colorValueB = (int)image[index + 2];
19
20            frequencyTableR[colorValueR]++;
21            frequencyTableG[colorValueG]++;
22            frequencyTableB[colorValueB]++;
23        }
24    }
25
26    // looping kedua untuk menghitung entropi
27    count = 0;
28    entropyR = 0;
29    entropyG = 0;
30    entropyB = 0;
31
32    for (int j = 0; j < height; ++j) {
33        for (int i = 0; i < width; ++i) {
34            int index = ((y + j) * absWidth + (x + i)) * channels;
35            int colorValueR = (int)image[index];
36            int colorValueG = (int)image[index + 1];
37            int colorValueB = (int)image[index + 2];
38
39            probabilityR = (double)frequencyTableR[count] / N;
40            probabilityG = (double)frequencyTableG[count] / N;
41            probabilityB = (double)frequencyTableB[count] / N;
42
43            entropyR += probabilityR * (log(probabilityR) / log(2.0));
44            entropyG += probabilityG * (log(probabilityG) / log(2.0));
45            entropyB += probabilityB * (log(probabilityB) / log(2.0));
46
47            count++;
48        }
49    }
50    entropyTotal = (-1) * (entropyR + entropyG + entropyB);
51
52    return entropyT
53 }
```

```

1 long getFileSize(const char* filePath) {
2     FILE* file = fopen(filePath, "rb"); // Membuka file dalam mode baca biner
3     if (file == NULL) {
4         fprintf(stderr, "Error: Tidak dapat membuka file %s\n", filePath);
5         return -1;
6     }
7
8     // Memindahkan pointer file ke akhir
9     fseek(file, 0, SEEK_END);
10
11    // Mendapatkan ukuran file
12    long fileSize = ftell(file);
13
14    // Menutup file
15    fclose(file);
16
17    return fileSize;
18 }

```

```

1 void DAC(int minimumBlockSize, QuadTree* root, unsigned char* image, unsigned char* newImage, int channels, int threshold, int absWidth, int method) {
2     attributes
3     int x = root->x;
4     int y = root->y;
5     int height = root->height;
6     int width = root->width;
7     int rootPixelSize = root->width * root->height; // ukuran blok pada root
8     double variance;
9
10    // Error Measurement Method
11    if (method == 1) { // variance normal
12        variance = calculate_variance(image, x, y, width, height, channels, root->colorMean, absWidth); // nilai variansi
13    } else if (method == 2) { // Mean Absolute Value
14        variance = calculate_MAD(image, x, y, width, height, channels, root->colorMean, absWidth); // nilai variansi
15    } else if (method == 3) { // Max Pixel Difference
16        variance = calculate_MP(image, x, y, width, height, channels, root->colorMean, absWidth); // nilai variansi
17        printf("MP\n");
18    } else if (method == 4) {
19        variance = calculate_Entropy(image, x, y, width, height, channels, root->colorMean, absWidth); // nilai variansi
20    }
21
22    // basis
23    if (rootPixelSize < minimumBlockSize || variance < threshold) {
24        generateImage(newImage);
25        for (int i = 0; i < height; ++i) {
26            for (int l = 0; l < width; ++l) {
27                int index = ((y + i) * absWidth + (x + l)) * channels;
28                newImage[index] = (unsigned char) (root->colorMean[0]);
29                newImage[index + 1] = (unsigned char) (root->colorMean[1]);
30                newImage[index + 2] = (unsigned char) (root->colorMean[2]);
31            }
32        }
33        // debug
34        // printf("Write at block %d, %d | pixel size : %d | colorMean : %d %d %d\n", x, y, rootPixelSize, root->colorMean[0], root->colorMean[1], root->colorMean[2]);
35    } else { // returns
36
37        QuadTree* firstChild = createNode(x, y, (width + 1)/2, (height + 1)/2, image, channels, absWidth);
38        QuadTree* secondChild = createNode(x + (width + 1)/2, y, width/2, (height + 1)/2, image, channels, absWidth);
39        QuadTree* thirdChild = createNode(x, y + (height + 1)/2, (width + 1)/2, height/2, image, channels, absWidth);
40        QuadTree* fourthChild = createNode(x + (width + 1)/2, y + (height + 1)/2, width/2, height/2, image, channels, absWidth);
41
42        connect(root, firstChild, 0);
43        connect(root, secondChild, 1);
44        connect(root, thirdChild, 2);
45        connect(root, fourthChild, 3);
46
47        DAC(minimumBlockSize, firstChild, image, newImage, channels, threshold, absWidth, method);
48        DAC(minimumBlockSize, secondChild, image, newImage, channels, threshold, absWidth, method);
49        DAC(minimumBlockSize, thirdChild, image, newImage, channels, threshold, absWidth, method);
50        DAC(minimumBlockSize, fourthChild, image, newImage, channels, threshold, absWidth, method);
51    }
52 }

```

Catatan : file header tidak dicantumkan pada snapshot kode karena sudah terwakilkan oleh file .c nya (dan alasan green computing 🌱)

BAB IV : Analisis Kompleksitas

Akan dilakukan analisis kompleksitas algoritma divide and conquer yang digunakan pada tugas kecil ini. Karena algoritma divide and conquer adalah algoritma dengan struktur yang rekursif, maka analisis dilakukan menggunakan Teorema Master yang sudah dipelajari di kelas.

Persamaan umum Teorema Master

$$T(n) = aT(n/b) + cn^d$$

Struktur pohon yang digunakan adalah Quadtree, yakni pohon dengan maksimal 4 anak. sehingga pada rekurens, fungsi/algoritma akan dipanggil ulang sebanyak 4 kali menyebabkan nilai **a = 4**. Kemudian pada setiap pemanggilan ulang (di rekurens), ukuran input berkurang menjadi $\frac{1}{4}$ aslinya menyebabkan nilai **b = 4**.

Masing-masing metode pengukuran error diukur kompleksitas algoritmanya, didapat hasil

- Variance method : **O(n) = n²**
- Mean Absolute Deviation : **O(n) = n²**
- Max Pixel Difference : **O(n) = n²**
- Entropy Method : **O(n) = n²**

Pada setiap pemanggilan pemanggilan fungsi/algoritma, salah satu metode di atas pasti dipanggil, oleh karena itu, nilai **d = 2**.

$$\begin{aligned} a &< b^d \\ 4 &< 4^2 \end{aligned}$$

Maka diperoleh hasil akhir

$$O(n) = n^2$$

Catatan : nilai n disini adalah max(width, height)

BAB V : Testing

No	Deskripsi Kasus	Output Terminal	Citra input	Citra output
1	Contoh kasus berhasil pada input file bertipe jpg. Metode : variance Threshold : 10 Blok minimum : 10	<pre>-----HASIL KOMPRESI----- Execution Time : 1.44 sec size before : 61.46 kB size after : 7.93 kB compression percentage : 9.48% Tree Depth : 8 Tree node count : 12999 Gambar berhasil diolah di D:\Repository\Tuc112_13523116\test\output.jpg</pre>		
2	Contoh kasus berhasil pada input file bertipe png. Metode : MAD Threshold : 10 Blok minimum : 10	<pre>-----HASIL KOMPRESI----- Execution Time : 1.02 sec size before : 20.23 kB size after : 7.73 kB compression percentage : 94.31% Tree Depth : 8 Tree node count : 15193</pre>		
3	Alamat absolut file input berisi file pdf	Error: Gagal memuat gambar di D:\Repository\Tuc112_13523116\test\liseornon.pdf PS D:\Repository\Tuc112_13523116\src	~	~
4	Masukan kosong	Masukkan alamat absolut file input >> Error: Alamat tidak boleh kosong.	~	~
5	Masukan bertipe beda	Masukkan threshold (integer) >> w01 Error: threshold harus berupa integer.	~	~
6	Masukan blok minimum tinggi Metode : Max Pixel Difference Threshold : 10 Blok minimum : 1000	<pre>-----HASIL KOMPRESI----- Execution Time : 0.53 sec size before : 61.46 kB size after : 36.64 kB compression percentage : 39.98% Tree Depth : 6 Tree node count : 1361</pre>		

<p>7 Masukan threshold tinggi</p> <p>Metode : Variance Threshold : 300 Blok minimum : 10</p>	<pre>=====HASIL KOMPRESI===== Execution Time : 1.06 sec size before : 61.45 kB size after : 54.85 kB compression percentage : 11.45% Tree depth : 10 Tree node count : 25841</pre>		
------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------	------------------------------------------------------------------------------------	-------------------------------------------------------------------------------------

BAB VI : 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	Mengimplementasikan 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	✓	

~ Pranala Github : [FityatulhaqRosyidi/Tucil2_13523116: Quadtree Image Compression](https://github.com/FityatulhaqRosyidi/Tucil2_13523116)

Referensi

[Algoritma Divide and Conquer](#)

<https://www.odelama.com/data-analysis/How-to-Compute-RGB-Image-Standard-Deviation-from-Channels-Statistics/>