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

Tugas Kecil 2 IF2211 Strategi Algoritma
Semester II tahun 2024/2025

Kompresi Gambar Dengan Metode Quadtree

by

William Andrian Dharma T
13523006

Github Repository Link : ***https://github.com/kirisame-ame/Tucil2_13523006***

I. Quadtree

Gambar 1. Contoh hasil kompresi 60%

Quadtree adalah suatu varian dari struktur data *tree* yang memiliki maksimal 4 *children* per *node* (bandingkan dengan binary tree yang memiliki maksimal 2 *children* per *node*). Pada kasus ini, Quadtree akan digunakan sebagai struktur data pembagi area gambar yang menyimpan :

1. Child nodes bertipe Quadtree : tl, tr, bl, br (*top left, top right, bottom left, bottom right*)
2. depth – kedalaman node dalam Quadtree
3. position : posisi (x,y) ujung kiri atas area gambar yang dimiliki
4. size : panjang dan lebar area gambar yang dimiliki
5. color : nilai RGB 8-bit rata-rata dalam area gambar yang dimiliki
6. isLeaf : boolean menandakan node berupa *leaf* atau bukan

Dengan Quadtree, sebuah gambar dapat dibuat lebih homogen dengan membuat pixel-pixel yang berwarna mirip berdasarkan suatu metrik tergabung menjadi suatu area yang berwarna sama, sehingga format penyimpan seperti png dan jpeg dapat lebih efisien mengompresi gambar yang membuat ukuran file lebih kecil.

II. Algoritma Kompresi

Kompresi Quadtree menggunakan konsep Divide and Conquer dan bekerja dengan membagi gambar menjadi 4 area jika area awal melewati batas homogenitas yang bergantung pada metrik yang digunakan. Contoh metrik yang dapat digunakan adalah Variance, Mean Absolute Deviation, Max Pixel Difference, Entropy, Structural Similarity Index (SSIM). Jika dianggap cukup dan ukuran area tidak lebih kecil dari *minimum block size* yang dipilih, maka akan dibuat 4 *child* yang merupakan area node Quadtree tersebut dibagi menjadi atas-kiri, atas-kanan, bawah-kiri, dan bawah-kanan, yang memiliki ukuran yang sama atau berbeda satu jika ukuran ganjil. Algoritma ini dipanggil secara rekursif pada *children* nya sebagai bagian dari Divide, dan pengecekan threshold adalah bagian Conquernya. Berikut prosedur dari pembuatan Quadtree ini :

1. Nilai-nilai RGB di area Quadtree diambil dan dihitung nilai rata-rata per-*channel*, dan disimpan sebagai atribut node.
2. Cek apakah area lebih kecil dari *minimum block size*, jika iya maka node Quadtree ini jangan dibagi lagi dan dijadikan *leaf*
3. Cek apakah area ini sudah cukup homogen berdasarkan metrik yang dipilih, jika tidak maka buat 4 *children nodes* hasil pembagian area sekarang, dan panggil algoritma pembuatan Quadtree pada semua *nodes* tersebut. Jika iya, maka Quadtree ini jangan dibagi lagi dan dijadikan *leaf node*.

Setelah pembuatan Quadtree, dilakukan proses Combine dari Divide and Conquer dengan cara memangil fungsi constructImage() dari *root* Quadtree. Fungsi tersebut bekerja dengan melakukan Depth First Search (DFS) dari Quadtree, dan mengisi sebuah array buffer sesuai dengan nilai atribut *color* dari root tersebut. Array buffer tersebut kemudian digunakan untuk membuat *image* sesuai format.

III. Analisis Divide and Conquer

Kompleksitas dari algoritma ini bernilai $O(NM\log_4(NM))$, dimana N adalah baris image dan M kolom image karena program terus didivide menjadi 4 subpersoalan kecil dimana perulangannya terus terbagi 4.

IV. Source Code Excerpts

```
 // ImageCompressor.cpp compressor main wrapper, starter
 1. void compressImage(const RunParams& runParams) {
 2.     int width = runParams.imageWidth;
 3.     int height = runParams.imageHeight;
 4.     RGB imgPix = runParams.imageBuffer;
 5.     array<int,3> meanColors = meanColor(imgPix);
 6.     if (width*height > runParams.minBlock && passThreshold(runParams, imgPix,meanColors,0,
width*height,runParams.threshold)) {
 7.         // Create the root node of the quadtree
 8.         unique_ptr<Quadtree> root = make_unique<Quadtree>(0, meanColors, array<int,2>{0,0},
array<int,2>{width,height});
 9.         // Build the quadtree
10.         root->buildQuadtree(runParams, 0, 0, width, height, 0,imgPix,runParams.threshold);
11.         // Save the compressed image
12.         string outputPath = runParams.outputPath;
13.         if (filesystem::exists(outputPath)) {
14.             cout << "File already exists. Overwriting..." << endl;
15.         }
```

```
16.         saveCompressedImage(root, outputPath, width,
height,runParams.extension,runParams.fileSize);
17.     } else {
18.         cout << "Image can't be compressed more." << endl;
19.     }
20.
21. }
22.
```

```
 // Quadtree.cpp Quadtree Builder function
 1. void Quadtree::buildQuadtree(const RunParams& runParams,int x,int y,int width,int height,int
depth,RGB& imgPix,double threshold) {
 2.     RGB pixels;
 3.     pixels[0].resize(width*height);
 4.     pixels[1].resize(width*height);
 5.     pixels[2].resize(width*height);
 6.     for (int i = 0; i < height; ++i) {
 7.         for (int j = 0; j < width; ++j) {
 8.             pixels[0][i*width+j] = imgPix[0][(y+i)*runParams.imageWidth+x+j];
 9.             pixels[1][i*width+j] = imgPix[1][(y+i)*runParams.imageWidth+x+j];
10.             pixels[2][i*width+j] = imgPix[2][(y+i)*runParams.imageWidth+x+j];
11.         }
12.     }
13.     array<int,3> meanColors = meanColor(pixels);
14.     this->color = meanColors;
15.     if (width * height <= runParams.minBlock|| width <= 1 || height <= 1) {
16.         this->isLeaf = true;
17.         this->tl = nullptr;
18.         this->tr = nullptr;
19.         this->bl = nullptr;
20.         this->br = nullptr;
21.         return;
22.     }
23.     if (passThreshold(runParams,
pixels,meanColors,y*runParams.imageWidth+x,width*height,threshold)) {
24.         this->isLeaf = false;
25.         this->tl = make_unique<Quadtree>(depth + 1, array<int,3>{0,0,0}, array<int,2>{x,y},
array<int,2>{width/2,height/2});
26.         this->tr = make_unique<Quadtree>(depth + 1, array<int,3>{0,0,0},
array<int,2>{x+width/2,y}, array<int,2>{width-width/2,height/2});
27.         this->bl = make_unique<Quadtree>(depth + 1, array<int,3>{0,0,0},
array<int,2>{x,y+height/2}, array<int,2>{width/2,height-height/2});
28.         this->br = make_unique<Quadtree>(depth + 1, array<int,3>{0,0,0},
array<int,2>{x+width/2,y+height/2}, array<int,2>{width-width/2, height-height/2});
29.         this->tl->buildQuadtree(runParams, x, y, width/2, height/2, depth +
1,imgPix,threshold);
30.         this->tr->buildQuadtree(runParams, x + width/2, y, width-width/2, height/2, depth +
1,imgPix,threshold);
31.         this->bl->buildQuadtree(runParams, x, y + height/2, width/2, height-height/2, depth
+ 1,imgPix,threshold);
32.         this->br->buildQuadtree(runParams, x + width/2, y + height/2, width-width/2, height-
height/2, depth + 1,imgPix,threshold);
33.     } else {
34.         this->isLeaf = true;
35.         this->tl = nullptr;
36.         this->tr = nullptr;
37.         this->bl = nullptr;
38.         this->br = nullptr;
39.     }
40. }
41.
```

```
 // Quadtree.cpp Quadtree to Image Buffer DFS
 1. void Quadtree::constructImage(unsigned char* output,int imgWidth, int *maxDepth,int
*nodeCount) {
 2.     if (maxDepth) {
 3.         *maxDepth = max(*maxDepth, depth);
 4.     }
 5.     if (nodeCount) {
 6.         (*nodeCount)++;
 7.     }
```

```
    if (isLeaf) {
        int x = position[0];
        int y = position[1];
        for (int i = 0; i < size[1]; ++i) {
            for (int j = 0; j < size[0]; ++j) {
                output[((y + i) * imgWidth + (x + j)) * 3] = color[0];
                output[((y + i) * imgWidth + (x + j)) * 3 + 1] = color[1];
                output[((y + i) * imgWidth + (x + j)) * 3 + 2] = color[2];
            }
        }
    } else {
        if (tl) tl->constructImage(output,imgWidth, maxDepth,nodeCount);
        if (tr) tr->constructImage(output,imgWidth, maxDepth,nodeCount);
        if (bl) bl->constructImage(output,imgWidth, maxDepth,nodeCount);
        if (br) br->constructImage(output,imgWidth, maxDepth,nodeCount);
    }
}

```

```
// ImageCompressor.cpp Compressed Image Saver
void saveCompressedImage(const unique_ptr<Quadtree>& root, const string& path,int width, int height,string extension,double origSize) {
    unsigned char* output = new unsigned char[width * height * 3];
    origSize /= 1024.0; // Convert to KB
    int maxDepth=0;
    int nodeCount = 0;
    root->constructImage(output, width, &maxDepth, &nodeCount);
    if (extension == ".png") {
        stbi_write_png(path.c_str(), width, height, 3, output, width * 3);
    } else if (extension == ".jpg" || extension == ".jpeg") {
        // Adjust quality based on original size and dimensions
        // Small memory sized images or large dimensions
        //tend to have bigger sizes on higher quality writes
        if (origSize < 100 || width*height >2000*2000) {
            stbi_write_jpg(path.c_str(), width, height, 3, output, 60);
        } else {
            stbi_write_jpg(path.c_str(), width, height, 3, output, 80);
        }
    } else if (extension == ".bmp") {
        stbi_write_bmp(path.c_str(), width, height, 3, output);
    } else {
        cout << "Unsupported file format" << endl;
    }
    double fileSize = filesystem::file_size(path) / 1024.0; // Convert to KB
    cout<<"Image dimensions: " << width << "x" << height << endl;
    cout<<"Original image size: " << origSize << " KB" << endl;
    cout << "Compressed image size: " << fileSize << " KB" << endl;
    cout<<"Compression Percentage: "<<(1-(fileSize/origSize))*100<<"%"<<endl;
    cout<<"Quadtree Depth: "<<maxDepth<<endl;
    cout<<"Node Count: "<<nodeCount<<endl;
    cout << "Compressed image saved to " << path << endl;
}

```

```
// Metrics.cpp BONUS SSIM
double ssim(const RGB& origImg,const array<int,3>& meanColors,int index,int size) {
    double c1 = 6.5025, c2 = 58.5225;

    double sumR = 0, sumG = 0, sumB = 0;
    double sumSqR = 0, sumSqG = 0, sumSqB = 0;

    for (int i = index; i < index + size; ++i) {
        sumR += origImg[0][i]; sumSqR += origImg[0][i] * origImg[0][i];
        sumG += origImg[1][i]; sumSqG += origImg[1][i] * origImg[1][i];
        sumB += origImg[2][i]; sumSqB += origImg[2][i] * origImg[2][i];
    }

    double meanR = sumR / size, meanG = sumG / size, meanB = sumB / size;
    double varR = (sumSqR - (sumR * sumR) / size) / size;
    double varG = (sumSqG - (sumG * sumG) / size) / size;
    double varB = (sumSqB - (sumB * sumB) / size) / size;
```

```
17.
18.     double redSSIM   = (2 * meanR * meanColors[0] + c1) * c2 / ((meanR * meanR +
meanColors[0] * meanColors[0] + c1) * (varR + c2));
19.     double greenSSIM = (2 * meanG * meanColors[1] + c1) * c2 / ((meanG * meanG +
meanColors[1] * meanColors[1] + c1) * (varG + c2));
20.     double blueSSIM  = (2 * meanB * meanColors[2] + c1) * c2 / ((meanB * meanB +
meanColors[2] * meanColors[2] + c1) * (varB + c2));
21.
22.     return (redSSIM + greenSSIM + blueSSIM) / 3.0;
23. }
24.
```

```
// View.cpp Abstracted API
1. // Program entry point wrapper function
2. void run(){
3.     startView();
4.     paramsInput();
5.     startCompression();
6. }
7.
```

```
  // Main.cpp Main wrapper
1. #include <iostream>
2. #include <View.hpp>
3. using namespace std;
4.
5. int main(){
6.     run();
7.     return 0;
8. }
```

V. Testing

File input testing terdapat pada directory assets/, hasil testing terdapat pada test/

A. Test case 1

Input:

kwkz.jpeg

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
1
Auto or Manual Thresholding (y/n)
n
Manual thresholding selected
Enter the threshold value for the quadtree compression
100
Enter the minimum block size for the quadtree compression
10
```

Output:

kwkz_compressed.jpeg

Metrics:

Result Metrics:

```
File already exists. Overwriting...
Image dimensions: 1103x1103
Original image size: 873.419 KB
Compressed image size: 202.337 KB
Compression Percentage: 76.8339%
Quadtree Depth: 9
Node Count: 106333
Compressed image saved to C:\Users\Warge\Documents\Kuliah\sem_4\stima\Tucil2_13523006/output/kwkz_compre
Compression completed in 2376 milliseconds
Quadtree compression completed
```

B. Test case 2

Input :

Saiba.jpeg

Output :

saiba_compressed.png

Metrics :

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
2
Auto or Manual Thresholding (y/n)
n
Manual thresholding selected
Enter the threshold value for the quadtree compression
10
Enter the minimum block size for the quadtree compression
10
```

Output Metrics:

```
Starting compression...
File already exists. Overwriting...
Image dimensions: 850x1063
Original image size: 296.033 KB
Compressed image size: 121.318 KB
Compression Percentage: 59.0187%
Quadtree Depth: 9
Node Count: 62445
Compressed image saved to C:\Users\Warge\Documents\Kuliah\sem_4\stima\Tucil2_
Compression completed in 597 milliseconds
Quadtree compression completed
```

C. Test Case 3

Input: hina.jpeg

Output: hina_compressed.jpeg

Metrics:

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
3
Auto or Manual Thresholding (y/n)
n
Manual thresholding selected
Enter the threshold value for the quadtree compression
30
Enter the minimum block size for the quadtree compression
20
```

Result Metrics:

```
Starting compression...
File already exists. Overwriting...
Image dimensions: 893x848
Original image size: 73.0293 KB
Compressed image size: 62.6865 KB
Compression Percentage: 14.1625%
Quadtree Depth: 8
Node Count: 29077
Compressed image saved to C:\Users\Warge\Documents\Kuliah\
Compression completed in 416 milliseconds
Quadtree compression completed
```

D. Test Case 4

Input: Hibiki.png

Output:
Hibiki_compressed.png

Metrics:

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
4
Auto or Manual Thresholding (y/n)
n
Manual thresholding selected
Enter the threshold value for the quadtree compression
1
Enter the minimum block size for the quadtree compression
30
```

Result Metrics:

```
Starting compression...
File already exists. Overwriting...
Image dimensions: 1100x1614
Original image size: 2176.41 KB
Compressed image size: 239.162 KB
Compression Percentage: 89.0112%
Quadtree Depth: 9
Node Count: 77809
Compressed image saved to C:\Users\Warge\Documents\Kuliah\
Compression completed in 1565 milliseconds
Quadtree compression completed
```

E. Test Case 5 (SSIM)

Input: steikfutsal.jpg

Output:

Steikfutsal_compressed.jpg

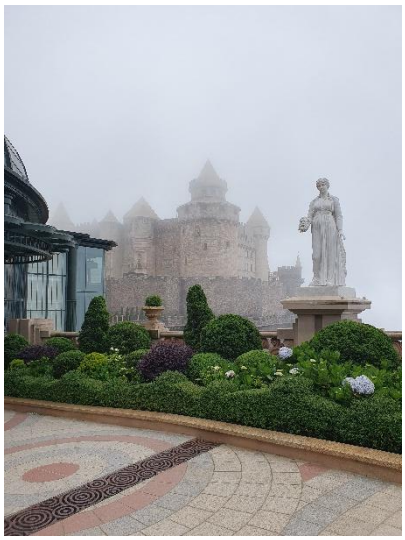
Metrics:

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
5
Auto or Manual Thresholding (y/n)
n
Manual thresholding selected
Enter the threshold value for the quadtree compression
0.7
Enter the minimum block size for the quadtree compression
20
```

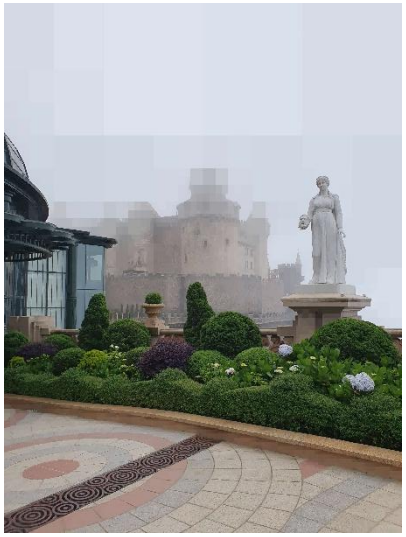
Result Metrics:

```
Starting compression...
Image dimensions: 1109x1479
Original image size: 475.354 KB
Compressed image size: 209.944 KB
Compression Percentage: 55.8341%
Quadtree Depth: 9
Node Count: 134977
Compressed image saved to C:\Users\Warge\Documents\Kuliah\
Compression completed in 1464 milliseconds
Quadtree compression completed
```

F. Test Case 6 (Auto-threshold MPD)
Input:
Castle.jpg



Output:
Castle_compressed.jpg



Metrics:

```
Saving in output directory
Enter the parameters for the quadtree compression
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
3
Auto or Manual Thresholding (y/n)
y
Auto thresholding selected
Enter the compression ratio (0-1) for the quadtree compression:
0.7
Compression ratio set to 70%
Enter the minimum block size for the quadtree compression
10
```

Result Metrics:

```
Best threshold: 63.75
Image dimensions: 4032x3024
Original image size: 3776.21 KB
Compressed image size: 800.447 KB
Compression Percentage: 78.8029%
Quadtree Depth: 11
Node Count: 684249
Compressed image saved to C:\Users\Warge\Documents\Kulia
Compression completed in 13034 milliseconds
Quadtree compression completed
```

G. Test Case 7 (Auto-threshold MAD)

Input :

kawachoco.png

Output : kawachoco_compressed.png

Metrics :

```
Enter the Error Metric you would like to use :
1. Variance
2. Mean Absolute Deviation
3. Max Pixel Difference
4. Entropy
5. Structural Similarity Index (SSIM) NOTE: Minimum Threshold
2
Auto or Manual Thresholding (y/n)
y
Auto thresholding selected
Enter the compression ratio (0-1) for the quadtree compression:
0.6
Compression ratio set to 60%
Enter the minimum block size for the quadtree compression
4
```

Output Metrics:

```
Compression Ratio too high, decreasing threshold...
[Threshold 3.98438] Compression Ratio: 0.600297
Best threshold: 3.98438
Image dimensions: 834x942
Original image size: 524.87 KB
Compressed image size: 209.792 KB
Compression Percentage: 60.0297%
Quadtree Depth: 9
Node Count: 69325
Compressed image saved to C:\Users\Warge\Documents\Kuliah\sem_
Compression completed in 2898 milliseconds
Quadtree compression completed
```

Tabel Pengerjaan

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	◯	