Penerapan Strategi Decrease and Conquer pada Kompresi Citra Grayscale melalui Pembagian Interval Probabilitas dengan Algoritma Arithmetic Coding

Dzaki Ahmad Al Hussainy - 13524084
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail: dzakiahmada7@gmail.com , 13524084@std.stei.itb.ac.id

***Abstrak*—Makalah ini menyajikan analisis penerapan kompresi citra *lossless* melalui kombinasi *Run-Length Encoding* (RLE) dan algoritma *Arithmetic Coding*. RLE diimplementasikan pada tahap pra-pemrosesan untuk mengurangi redundansi spasial pada piksel-piksel yang bernilai sama. Selanjutnya, algoritma *Arithmetic Coding* berbasis pendekatan *Decrease and Conquer* digunakan untuk memampatkan data secara optimal melalui pembagian interval probabilitas. Performa metode gabungan ini dievaluasi menggunakan dataset citra *grayscale* standar. Kinerja algoritma dianalisis berdasarkan metrik standar kompresi, dengan fokus utama pada perolehan Rasio Kompresi (CR).**

Kata Kunci*— *Arithmetic Coding; Decrease* and Conquer*; kompresi* lossless*;* grayscale*;* Run-Length Encoding *(RLE).

I. PEMBUKAAN

Pesatnya perkembangan teknologi digital telah meningkatkan kebutuhan akan metode penyimpanan dan transmisi data yang efisien, khususnya pada citra digital. Kebutuhan akan kompresi *lossless* menjadi krusial dalam domain yang sensitif terhadap integritas data, seperti citra medis dan sistem pengarsipan digital. Citra *grayscale* umumnya direpresentasikan oleh satu nilai intensitas per piksel dan sering kali memiliki area dengan intensitas yang seragam. Karakteristik ini memungkinkan penggunaan *Run-Length Encoding* (RLE) sebagai metode yang sederhana dan efisien untuk mereduksi redundansi spasial dengan mengodekan run dari piksel-piksel yang identik.

Untuk meningkatkan efisiensi kompresi, RLE dapat dikombinasikan dengan algoritma *Arithmetic Coding*. Pendekatan *Arithmetic Coding* yang berbasis paradigma *decrease and conquer* menawarkan mekanisme kompresi yang optimal melalui pembagian interval probabilitas secara rekursif. Makalah ini menyajikan evaluasi kinerja kompresi citra *grayscale* dengan metode gabungan RLE dan *Arithmetic Coding*. Analisis efisiensi dilakukan berdasarkan metrik rasio kompresi (CR), dan hasil yang diperoleh dibandingkan dengan metode kompresi RLE-Huffman untuk mengukur keunggulan performa teknik yang diusulkan.

II. DASAR TEORI

A. *Citra Digital*

Dalam prosesan citra, citra didefinisikan dengan sebuah gambar yang berada di dua dimensi atau dwimatra. Citra dapat direpresentasikan secara matematis sebagai fungsi f(x,y) dengan x dan y mewakili koordinat dua dimensi, sedangkan fungsi f menunjukan tinggat intensitas cahaya pada titik tertentu [1].

Citra dapat diklasifikasi berdasarkan jumlah *frame* yang dimilikinya. Secara umum ada dua jenis. Citra statik atau diam dan citra bergerak (gambar). Citra statik hanya memiliki satu *frame* yang tidak berubah dengan waktu. Berbeda dengan citra bergerak yang memiliki urutan *frame* yang ditampilkan dengan cepat untuk memberikan ilusi gambar bergerak. *Frame* pada gambar bergerak secara umum adalah citra yang tertangkap pada waktu tertentu [1].

Sebuah citra gambar adalah representasi diskrit dari gelombang cahaya dalam sebuah dimensi ruang dan waktu [1]. Mengambil sampel di dimensi waktu adalah proses mengambil grame dalam sebuah waktu dan membuat video sedangkan mengambil sample di dimensi ruang mengambarkan mengambil intensitas cahaya secara diskrit pada titik (x,y) dalam sebuah *frame* [1].

Citra digital biasanya direpresentasikan dalam matriks dua dimensi dengan ukuran M x N dengan M dan N mengambarkan resolusi dari sebuah citra.

$$f(x,y) = \begin{bmatrix} f(0,0) & \cdots & f(0,N-1) \\ \vdots & \ddots & \vdots \\ f(M-1,0) & \cdots & f(M-1,N-1) \end{bmatrix}$$

Masing-masing elemen di matriks mengambarkan piksel (*picture element*) yang menyimpan intensitas atau warna pada lokasi ruang spesifik [1]. Masing-masing piksel dalam sebuah citra didefinisikan dengan fungsi f(x,y) yang mengambarkan

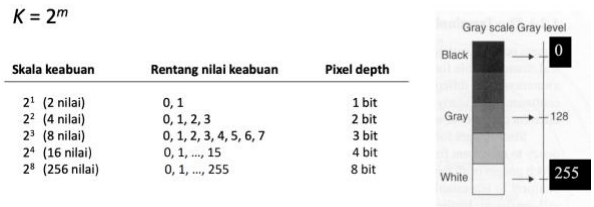
intesitas kecerahan atau kegelapan dengan sebuah koordinat (x,y). Contohnya **Gambar 2.A.1** memiliki resolusi 1200 x 1500 piksel yang artinya memiliki 1800000 individual pixel. Artinya gambar memiliki 1200 baris dan 1500 kolom dan masing-masing piksel memiliki intensitas atau warna pada posisi (x,y) tertentu pada gambar tersebut.

Gambar 2.A.1 Ilustrasi piksel pada citra
(Sumber : [1])

Sebuah nilai dari f(x,y) itu tergantung dari tipe sebuah citra. Dalam citra *grayscale,* f(x,y) memiliki satu nilai yang mengambarkan kecerahan biasanya memiliki jangkauan dari hitam sampai putih. Lebih umumnya pada gambar berwarna menggunakan fungsi yang menghasilkan tuple warna yaitu f(x,y) = (R,G,B). Masing-masing titik akan menghasilkan intensitas sebuah warna merah, hijau, dan biru yang akan merepresentasikan sebuah gambar di dunia nyata.

B. *Citra Gray Scale*

Citra *grayscale* adalah citra yang merepresentasikan intensitas cahaya berdasarkan tingkat kuantisasi cahaya [2]. Kuantisasi cahaya berdasarkan proses membuat rentang cahaya kontigu dan dipecah-pecah dijadikan disrkit untuk masing-masing koordinat (x,y) [2]. Tujuan dari kuantisasi ini adalah untuk mempetakan cahaya yang kontigu menjadi k banyaknya tingkat cahaya. Masing-masing tingkat cahaya dari sebuah citra di representasikan dengan rentang [0, k - 1] dengan rentang ini adalah tingkat intensitas keabuan. Kuantisasi ini memecah intensitas cahaya menjadi rentang k dengan nilai $0, 1, 2, \ldots,\ \mathrm{k}-1$ [2]. Dengan biasanya k dipilih dari nilai berkelipatan 2 yang diekspresikan dengan $k = 2^m$, dengan k mendefinisikan banyaknya tingkat keabuan yang ada dan m memberikan banyaknya bit dalam komputer yang digunakan untuk merepresentasikan sebuah piksel [2].



Gambar 2.B.1 representasi dari nilai K
(Sumber : [2])

C. *Kompresi Citra*

Kompresi citra adalah sebuah teknik yang digunakan untuk mengurangi redudansi pada representasi data dengan tujuan meminimalisasi penyimpanan data tanpa mengurangi kualitas citra. Citra yang sudah dikompresi membutuhkan lebih sedikit penyimpanan dibanding citra yang tidak dikompresi sehingga transmisi data yang dikompresi biasanya lebih cepat dan lebih efisien [3].

Tujuan utama dari kompresi citra adalah untuk mengeliminasi redundansi dan data yang repetitif sehingga bisa direpresentasikan lebih padat. Data yang redunan di sini adalah pola atau elemen pada gambar yang berulang dan dapat dikodekan dengan lebih effisien. Ada tiga jenis utama dari data redundan dalam citra.

1. Coding Redundancy
   Terjadi ketika sebuah hasil pengkodean dari piksel yang sama direpresentasikan dengan data yang lebih panjang atau yang tidak efisien [3].
2. Spacial/Temporal Redundancy
   Terjadi ketika sebuah piksel yang bertentangga dalam sebuah video dalam dimensi ruang atau dalam dimensi waktu yang memiliki nilai yang sama atau memiliki intensitas yang sama [3].
3. Psychovisual Redundancy
   Terjadi ketika ada sebuah informasi yang dipersepsikan tidak lebih penting atau bahkan tidak dapat terlihat oleh mata manusia [3].

D. *Metode Kompresi Citra*

Metode dalam melakukan kompresi citra dapat dibagi menjadi dua jenis berdasarkan bagaimana cara algoritma menyimpan semua data semuanya.

1. *Lossy Compression*
   Kompresi *lossy* menghasilkan citra hasil pemampatan yang hampir sama dengan citra semula namun ada informasi yang hilang akibat pemampatan tetapi dapat ditolerir oleh persepsi visual [3]. Kompresi ini umumnya mendapatkan rasio kompresi lebih mangkus dibanding teknik lossless dan biasanya digunakan pada aplikasi yang tidak mengutamakan keakuratan seperti gambar pada web dan foto digital. Contoh umum dari penggunaan kompresi ini ada pada kompresi pada JPEG dan kompresi citra berbasis fraktal [3].

2. *Lossless Compression*
   Kompresi *lossless* menghasilkan hasil pemampatan yang tepat sama dengan citra semula, pixel per pixel dan tidak ada informasi yang hilang akhibat pemampatan. Umumnya nisbah pemampatan rendah namun kualitas citra mampat tetap tinggi. Penggunaan pemampatan *losses* ini digunakan pada citra yang tidak boleh terdegradasi akibat pemampatannya misalnya citra medis dan citra x-ray [3]. Contoh umumnya adalah algoritma huffman, *run-length encoding* dan *Arithmetic Coding*.

E. Algoritma *Run-Length Encoding*

Run-length Encoding (RLE) adalah teknik pemampatan yang digunakan untuk mengurangi uluran citra yang memiliki intensitas sebuah gambar yang sama [3]. Cara bekerjanya

adalah membuat pasangan (p, q) dengan p adalah intensitas cahaya yang sama dan q adalah banyaknya piksel yang memiliki intensitas yang sama. Penjumlahan pasangan ini disebut dengan *run length.*

Contohnya misalnya sebuah baris pada citra *grayscale* memiliki [120, 120, 120, 45, 45] maka akan dikodekan menjadi [(120, 3), (45,2)]. Pengodean ini membantu untuk menghilangkan redudansi nilai urutan pixel yang sama.

F. Strategi *Decrease and Conquer*

Strategi *Decrease and Conquer* merupakan paradigma desain algoritma yang mereduksi sebuah persoalan berukuran besar menjadi hanya satu upa-persoalan (*sub-problem*) yang berukuran lebih kecil. Berbeda dengan *Divide and Conquer* yang membagi dan memproses seluruh upa-persoalan, strategi *Decrease and Conquer* hanya berfokus pada pemrosesan satu upa-persoalan secara linier dan mengabaikan sisa bagian lainnya setelah reduksi terjadi [4].

Algoritma *arithmetic coding* ini menggunakan variasi *decease-by-constant* dengan konstanta 1. Definisi formal dari strategi ini dalam konteks pemampatan data adalah sebagai berikut.

Misalkan sebuah untaian data (string) S dengan panjang karakter n karakter, dengan $S = \{s_1, s_2, \ldots, s_n\}$. Proses pengodean akan mereduksi masalah berukuran n-1 dengan mengambil simbol pertama s_1 untuk dikodekan kedalam inteval biner, kemudian mengulangi prosedur yang sama (rekursif) secara linier untuk sisa untai data $S' = \{s_2, s_3, \ldots, s_n\}$ hingga untaian mencapai kasus dasar (base case) yaitu n = 0.

Penerapan strategi ini berjalan beriringan dengan penyusutan ruang pencarian (*search space*) pada garis bilangan secara bertahap yaitu setiap iterasi penurutan karakter akan mempersempit batas interval secara konstan.

G. Algoritma *Arithmetic Coding*

Algoritma *Arithmetic Coding* adalah teknik kompresi peka-konteks (*context-sensitive*) berkinerja tinggi yang merepresentasikannya seluruh untai data ke dalam satu nilai pecahan biner tunggal yang berada di dalam rentang interval [0, 1). Algoritma ini bekerja secara rekursif dengan melakukan partisi interval secara relatif berdasarkan distribusi probabilitas dari masing-masing simbol yang menyusun data tersebut [5].

1) Komponen Utama Algoritma

Secara umum algoritma ini akan memiliki

1. Struktur Model
   Sebuah mesin kondisi terbatas yang melacak kondisi lingkungan (*context*) dari kemunculan simbol, memastikan pemetaan rentang interval bersifat dinamis dan adaptif terhadap karakteristik spasial data citra.
2. Estimasi Statistik
   Komponen yang bertanggung jawab menghitung nilai probabilitas asli (p) serta probabilitas akumulatif (P) dari setiap simbol untuk menentukan koordinat awal dan lebar dari partisi garis bilangan.
3. Encoder
   Unit pemampat (*encoder*) yang menerima masukan data mentah dan menghasilkan representasi biner akhir, serta unit pengurai (*decoder*) yang merekonstruksi kembali nilai biner tersebut menjadi data citra asal melalui operasi penskalaan ulang (*rescaling*).

2) Formulasi Matematis Rekursi Inteval

Secara formal, opereasi penyusutan interval pada setiap tahapan rekursi *decrease and conquer* dikendalikan oleh dua variable utama, yaitu *Code Point (C)* yang merepresentasikan batas bawah interval, dan *Interval Width (A)* yang merepresentasikan lebar ruang pencarian saat itu [5].

Misalkan pada iterasi ke-*k*, simbol yang diproses adalah S_k dengan nilai probabilitas akumulatif P(S_k) dan probabilitas biner p(S_k). Maka pembaruan parameter interval ditentukan melalui persamaan matematika berikut:

$$A_k = A_{k-1} \times p(S_k)$$

$$C_k = C_{k-1} + (A_{k-1} \times P(S_k))$$

Nilai $(A_{k-1} \times P(S_k))$ pada persamaan di atas didefinisikan sebagai *augmend* [5], yaitu nilai pergeseran skala pecahan yang ditambahkan ke dalam basis kode string sebelumnya. Kondisi awal (*inisial state*) sebelum interasi pertama dimulai diset pada nilai mutlak garis bilangan dasar [5].

$A_0 = 1.0_2$ dan $C_0 = 0.0_2$

3) Formulasi Matematis Rekursi Inteval

Sebelum proses pengkodean dilakukan, algoritma melakukan analisis frekuensi terhadap data uji untuk menyusun metrik probabilitas.

Simbol	Probilitas Akumulatif (P)	Probabilitas Asli	Probabilitas dalam Biner (p)	Panjang bit
d	0.000	1/8	.001	3
b	0.001	1/4	.01	2
a	0.011	1/2	.1	1
c	0.111	1/8	.001	3

Tabel 2.G.1 Hasil Perhitungan Statistik Arithmatic Coding

4) Implementasi Pengkodean

Sebagai ilustrasi pengeksekusian algoritma, dilakukan eksperimen pemampatan terhadap untai data sampel "aabc" menggunakan model statistik pada **Tabel 2.G.1**. Proses *Decrease and Conquer* berjalan melalui tahapan reduksi sebagai berikut:

Iterasi 1 (Simbol 'a') :

Mengurangi masalah menjadi sisa string "abc". Melakukan kalkulasi berdasarkan parameter $P('a') = 0.011_2$ dan $p('a') = 0.1_2$

$$C_1 = 0 + (1 \times 0.011_2) = 0.011_2$$
$$A_1 = 1 \times 0.1_2 = 0.1_2$$

Interval terkini menyusut menjadi $[0.011_2, 0.111_2)$.

Iterasi 2 (Simbol 'a') :
Mengurangi masalah menjadi sisa string "bc". Melakukan kalkulasi berdasarkan parameter $P('a') = 0.011_2$ dan $p('a') = 0.1_2$

$$C_2 = 0.011_2 + (0.1_2 \times 0.011_2) = 0.1001_2$$
$$A_2 = 0.1_2 \times 0.1_2 = 0.01_2$$

Interval terkini menyusut menjadi $[0.101_2, 0.1101_2)$.

Iterasi 3 (Simbol 'b') :
Mengurangi masalah menjadi sisa string "c". Melakukan kalkulasi berdasarkan parameter $P('b') = 0.001_2$ dan $p('b') = 0.01_2$

$$C_3 = 0.1001_2 + (0.1_2 \times 0.001_2) = 0.10011_2$$
$$A_3 = 0.01_2 \times 0.01_2 = 0.0001_2$$

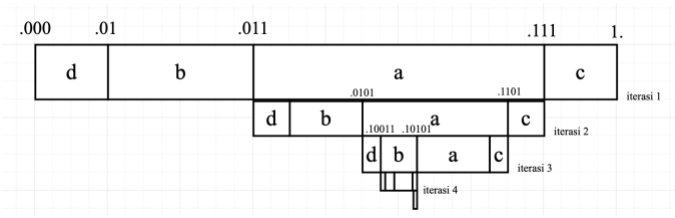
Interval terkini menyusut menjadi $[0.10011_2, 0.10101_2)$.

Iterasi 4 (Simbol 'c') :
Sisa stirng habis ukuran string = 0. Melakukan kalkulasi berdasarkan parameter $P('c') = 0.111_2$ dan $p('c') = 0.001_2$

$$C_4 = 0.10011_2 + (0.0001_2 \times 0.111_2) = 0.1010011_2$$
$$A_4 = 0.0001_2 \times 0.001_2 = 0.0000001_2$$

Interval terkini menyusut menjadi $[0.1010011_2, 0.1010100_2)$.

Melalui pencapaian hingga kasus dasar (base case), string "aabc" berhasil direduksi penuh menjadi satu titik kode tunggal, yaitu 1010011. Operasi matematis ini dapat diimplementasikan secara efisien pada tingkat perangkat keras dengan mengantu fungsi perkalian fraksional menggunakan operasi bit ke kanan (bitwise right shift) yang sebanding dengan nilai eksponensial basis 2 dari lebar interval sebelumnya.



Gambar 2.G.1 Ilustrasi pengkodean
Sumber : Arsip penulis

H. *Pendekatan Decrease and Conquer Pada Pemampatan Citra*

Dekomposisi algoritma ini akan membaginya menjadi beberapa tahap sebagai berikut

1) Decrease Phase

Pada fase ini permasalahan citra berupa matriks berukuran M x N piksel disusutka skalanya. Citra tidak akan dibaca sebagai satu kesatuan aluran data linier, melainkan akan dipartisi menjadi blok-blok kecil berukuran 8 x 8 piksel [7]. Alasan dipilih ukran blok ini karena sudah terbukti efisien. Langkah pemecahan ini berfungsi untuk membatasi ruang lingkup observasi algortima *arithmetic coding* hanya pada 64 piksel dalam satu waktu. Secara teretis, fase decrease ini secara langsung memecahkan masalah tinggi keacakan lokal data menjadi piksel-piksel di dalam radius blok 8 x 8 yang memiliki korelasi spasial yang kuat dan cendrung homogen.

2) Conquer Phase

Setelah masalah direduksi menjadi blok-blok kecil algoritma akan memanggil dua algoritma kompresi dijalankan. Run-lengh Encoding kemudian melakukan Arithmetic coding. Karena variasi simvol dalam blok 8 x8 sangat sedikit, tabel frekuensi loakl yang terbentuk akan menghasilkan probabilitas (p) yang tinggi untuk setiap simbol. Algoritma arithmetic coding dapat menyelesaikan data ini dengan mudah karena interval probabilitas tidak akan menyusut secara ekstrem sehingga kode akhir dapat direpresentasikan menggunakan jumlah bit pecahan yang sangat minimum. Setelah melakukan perhitungan tabel frekuensi akan dibuat sistem BitStreamWriter yang akan menuliskan potongan data tersebut kedalam satu file biner secara sekuensial yang berisi potongan sebagai berikut tabel frekuensi lokal milik blok tersebut dan *payload* bit frakisonal hasil *arithmetic coding*

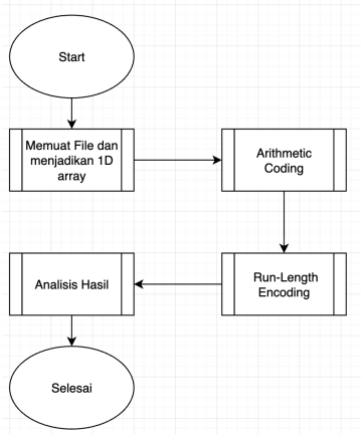
III. IMPLEMENTASI

A. *Batasan dan Cakupan Analisa*

Makalah ini hanya akan berfokus pada analisa citra *grayscale* dengan intensitas dari 0 (hitam) hingga 255 (putih) yaitu dengan mengunakan nilai k = 8 bits. Pada Makalah ini juga tidak akan membahas lebih dalam mengenai meta data citra, header file atau format spesifik sebuah citra.

B. *Arsitektur Program*

Arsitektur yang digunakan pada program ini adalah *pipe line* karena hanya melakukan pemerosesan input dan output. Program ini memiliki alur sebagai berikut.



Gambar 3.B.1 *Flow Chart* Arsitektur Program
Sumber : Arsip penulis

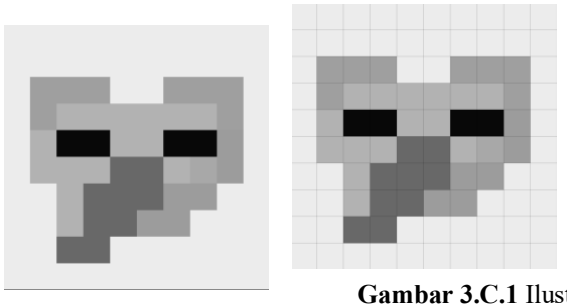
Pada tahap awal pemrosesan, citra *grayscale* dimuat oleh program dan dilinearisasi menjadi sebuah *array* satu dimensi menggunakan metode pemindaian horizontal (*raster scan*). Data linier tersebut kemudian diproses menggunakan metode *Run-Length Encoding* (RLE) untuk mereduksi deretan piksel berdekatan yang memiliki intensitas identik. Keluaran dari tahap ini berupa tupel pasangan data yang memuat nilai intensitas dan jumlah perulangannya (*count*). Selanjutnya, masing-masing pasangan RLE tersebut diproses menggunakan algoritma *Arithmetic Coding* dengan pendekatan *Decrease and Conquer*. Untuk menjaga akurasi kompresi, diterapkan model probabilitas ganda (*dual-model*), di mana pembaruan batas bawah dan lebar interval pada garis bilangan dieksekusi secara terpisah namun berkesinambungan untuk masing-masing nilai intensitas dan nilai *count* berdasarkan distribusi statistik spasialnya.

C. *Tahap Pembacaan Citra*

Tahapan awal dimulai dengan membaca file citra digital mengunakan pustaka open-source `stb_image.h` .

```
unsigned char *img =
stbi_load(input_image.c_str(), &width, &height,
&channels, 1);
```

Fungsi ini membuat data biner citra dari media penyimpanan kedalam memori RAM berupa pointer array 1 dimensi berurutan (`unsigned char *`). Parameter angka 1 terakhir ini mekasa citra dikonversi langsung ke format *grayscale* (8-bit).



Gambar 3.C.1 Ilustrasi 10x10 piksel citra *grayscale.*

(Sumber: arsip penulis)

Ilustrasi yang mirip gajah ini dapat direpresentasikan ke dalam matriks 2D dengan masing-masing elemen berkoresponden dengan tingkat keabuannya. Matriks ini yang merepresentasikan citra untuk melakukan proses selanjutnya.

$$\begin{bmatrix} 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 \\ 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 \\ 236 & 159 & 159 & 159 & 236 & 236 & 159 & 159 & 159 & 236 \\ 236 & 159 & 178 & 178 & 178 & 178 & 178 & 178 & 159 & 236 \\ 236 & 178 & 8 & 8 & 178 & 178 & 8 & 8 & 156 & 236 \\ 236 & 178 & 178 & 178 & 104 & 104 & 178 & 168 & 156 & 236 \\ 236 & 236 & 178 & 104 & 104 & 104 & 156 & 156 & 236 & 236 \\ 236 & 236 & 178 & 104 & 104 & 156 & 156 & 236 & 236 & 236 \\ 236 & 236 & 104 & 104 & 236 & 236 & 236 & 236 & 236 & 236 \\ 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 & 236 \end{bmatrix}$$

Representasi matriks di atas akan diproses berdasarkan kesamaan lokalitas dengan ukuran blok yang akan dibahas di tahap selanjutnya.

D. *Ekstrasi Blok Citra*

Untuk meningkatan efisiensi dan memanfaatkan redundansi lokal, citra tidak diproses sekaligus, namun dipecah menjadi blok-blok spasial berukuran 8×8 piksel menggunakan fungsi `extract_linier_block`.

```
std::vector<int> extract_linear_block(unsigned char *img, int start_x, int
start_y, int img_width, int img_height) {
        std::vector<int> block_1d;
        block_1d.reserve(block_size * block_size);
        for (int y = 0; y < block_size; y++) {
            for (int x = 0; x < block_size; x++) {
                if (start_x + x < img_width && start_y + y < img_height) {
                    int pixel_idx = ((start_y + y) * img_width) + (start_x + x);
                    block_1d.push_back(img[pixel_idx]);
                }
            }
        }
        return block_1d;
}
```

Dalam fungsi ini menggunakan pengulangan bersarang secara spasial sejauh rentang `block_size` (8 piksel). Fungsi ini memetakan koordinat matriks 2D (x,y) kembali ke indeks linier array 1D asli melalui rumus index

```
int pixel_idx = ((start_y + y) * img_width) + (start_x + x);
```

Digunakan juga penanganan batas untuk memastika citra tidak kelipatan 8, koordinat diluar batas gambar tidak akan diakses. Hasilnya akan dimasukan ke dalam vektor `block_1d` berkapasitas 64 elemen.

E. *Kompresi Tahap Pertama (RLE)*

Vektor piksel 1D hasil ekstraksi blok kemudian dialirkan ke fungsi apply_rle untuk mereduksi redundansi data yang

berurutan secara beruntun. Di sini diimplementasikan algoritma standar RLE yang akan menghasilkan struktur data `RLE_Pair` yang berisi pasangan `{intensity, count}`. Tahapan ini secara drastis mengurangi simbol yang harus diproses pada tahap berikutnya.

```
std::vector<RLE Pair> apply rle(const std::vector<int> &data) {
        std::vector<RLE_Pair> rle_data;
        if (data.empty()) return rle data;

        int n = data.size();
        for (int i = 0; i < n; i++) {
            int count = 1;
            while (i < n - 1 && data[i] == data[i + 1]) {
                count++;
                i++;
            }
            rle data.push back({data[i], count});
        }
        return rle_data;
    }
```

F. Pemodelan Statistik

Sebelum dilakukan kompresi *arithmetic* aliran data RLE harus dianalisis karekteristik statistikanya melalui kelas `ProbabilityModel`. Kelas ini dibangun berdasarkan simbol RLE bukan piksel mentah. Alur dari kelas ini adalah sistem akan menghitung seberapa sering suatu nilai intensitas muncul di dalam RLE stream pada blok tersebut dan disimpan dalam `std::map<int, int> frequency`. Kemudian berdasarkan frekuensi blok ini akan dihitung distribusi probabilitas individu (*p*) dengan membagi frekuensi simbol dibagi dengan total simbol RLE (`total_symbol`) dan bersamaan dengan itu dihitung probabilitas kumulatif (P) yang bergungsi sebagai batas bawah (*lowerbound*) interval dalam *arithmetic coding.*

```
struct SymbolProb {
    double p; // Probabilitas Asli
    double P; // Probabilitas Akumulatif
};

// KELAS PROBABILITY MODEL
class ProbabilityModel {
private:
    std::map<int, int> frequency;
    std::map<int, SymbolProb> prob_table;

public:
    ProbabilityModel(const std::vector<RLE Pair> &rle stream)
    {
        int total_symbol = rle_stream.size();
        for (const auto &item : rle_stream) {
            frequency[item.intensity]++;
        }

        double cumulative = 0.0;
        for (const auto &pair : frequency) {
            int symbol = pair.first;
            double prob = (double)pair.second / total_symbol;

            prob_table[symbol] = {prob, cumulative};
            cumulative += prob;
        }
    }
    std::map<int, int> get frequencies() const { return frequency; }
    std::map<int, SymbolProb> get_probability_table() const { return prob_table;
}
};
```

G. Kompresi Tahap Akhir Arithmetic Coder

Tahapan akhir dalam mereduksi bit menggunakan kelas `ArithmeticCoder` melewati fungsi encode. Untuk setiap simbol RLE yang dibaca, algortima secara dinamis memperbaharui interval akhir (C) dan mempersempit rentang interval yang tersedia (A) berdasarkan aturan matermatika yang dijelaskan di bab sebelumnya. Berbedaanya di sini digunakan secara iteratif agar menggurangi pemanggilan fungsi yang dapat menghabiskan stack.

```
class ArithmeticCoder {
public:
    double encode(const std::vector<RLE Pair> &rle stream, const
ProbabilityModel &model)
    {
        auto prob_table = model.get_probability_table();
        double A = 1.0;
        double C = 0.0;

        for (size t i = 0; i < rle stream.size(); i++) {
            int current_symbol = rle_stream[i].intensity;
            SymbolProb prob = prob table[current symbol];

            C = C + (A * prob.P);
            A = A * prob.p;
        }
        return C;
    }
};
```

Output akhir seluruh simbol RLE dalam bentuk satu blok selesai diproses, fungsi mengembalikan nilai `double` akhir (final code point C) yang merepresentasikan blok 8x8 tersebut secara aman dan padat.

H. Implementasi Utama Program

```
void compress(const std::string &input_image, const std::string &output_bin) {
        int width, height, channels;
        unsigned char *img = stbi_load(input_image.c_str(), &width, &height,
&channels, 1);

        if (!img) return;

        double total ac bits = 0;
        double total_count_bits = 0;

        try {
            {
                BitstreamWriter writer(output_bin);
                writer.write_global_header(width, height, block_size);
                ArithmeticCoder coder;

                for (int y = 0; y < height; y += block size) {
                    for (int x = 0; x < width; x += block_size) {
                        std::vector<int> block ld = extract linear block(img, x,
y, width, height);
                        std::vector<RLE_Pair> rle_stream = apply_rle(block_ld);

                        ProbabilityModel model(rle_stream);
                        double final_code_point = coder.encode(rle_stream,
model);

                        BlockData data_to_save;
                        data_to_save.frequencies = model.get_frequencies();
                        data_to_save.final_code_point = final_code_point;
                        writer.write block(data to save);

                        total count bits += (rle stream.size() * 8.0);

                        double block A = 1.0;
                        auto prob_table = model.get_probability_table();
                        for (const auto &item : rle_stream) {
                            block_A *= prob_table[item.intensity].p;
                        }

                        if (block_A > 0 && block_A < 1.0) {
                            total ac bits += std::ceil(-std::log2(block A));
                        }
                    }
                }
            }

. . . Potongan Program utama
```

Alur program ini akan melakukan memuat file dan melakukan inisalisasi program kemudian akan menuliskan header dari filenya dan melakukan kompresi secara iteratif blok per blok dan melakukan pemerosesan data setiap grid 8×8 piksel kemudian diakhir akan dituliskan kedalam file biner.

I. Perhitungan Compression Ratio

Dalam melakukan evaluasi metode ini diperlukan untuk melakukan evaluasi ukuran file ukuran awal dan kemudian membandingkan setelah dilakukan pemampatan. Perhitungan ini dapat di hitung dengan rumus

$$Compression\ ratio\ (C_r) = \left(\frac{Compression\ size}{Original\ Size}\right) \times\ 100\ \%$$

Dengan Cr adalah compression ratio (Rasio pemampatan dalam persen), Original Size adalah ukuran citra asli sebelum dikompresi (8 bit × lebar × tinggi) dan Compressied size adalah ukuran data hasil kompresi.

Dalam Makalah ini compression rasio akan dievaluasi menggunakan dua skenario analisis

1) Pengukuran berdasarkan ukuran nyata

Pada skenario ini *compressed sized* merujuk pada ukuran total biner (.bin) yang tersimpan di dalam media penyimpanan (disk). Nilai ini mencakup keseluruhan struktur data arsitektur file kompresi yang terdiri dari

1. *Global Header* berisi informasi dimensi citra dan ukuran blok
2. *Local Header* berisi Tabel frekuensi kemunculan simbol piksel dan jumlah simbol unik yang wajib disimpan agar proses dekompresi dapat dilakukan
3. Payload kompresi berisi nilai titik kode pecahan hasil proses *arithmetic coding*.

2) Pengukuran berdasarkan ukuran teoritis

Pada skenario ini *Compressed sized* dihitung secara matematis menggunakan teori informasi shannon (Entropi) untuk mengetahui batas kemampuan optimal dari algoritma tanpa dipengaruhi oleh defisit ukuran akibat penyimpanan metadata file format. Di sini ukuran hanya dihitung berdasarkan akumulasi bit murni

$$Compressed\ Size_{teoretis} = Bit\ RLE + Bit\ AC$$

Bit RLE adalah jumlah representasi bit dari panjang barisan (*run length*) piksel kembar (diasumsikan secara teoretis konstan sebesar 8 bit per pasangan) dan bit AC (*Arithmetic Coding*) adalah jumlah bit fraksional yang dibutuhkan untuk mengodekan nilai piksel yang dihitung dari penciutan rentang intercal probabilitas akhir (A)melalui hukum algoritma $\lceil -\log_2(A) \rceil$ [5].

IV. HASIL DAN ANALISIS

Setelah melakukan analisis dengan contoh pada **Gambar 3.C.1** Didapatkan hasil berikut:

```
==========================================
            LAPORAN KOMPRESI CITRA 8x8
==========================================

[1] METRIK DUNIA NYATA (DENGAN METADATA / UKURAN FILE)
------------------------------------------------
Ukuran Citra Asli     : 100 Byte
Ukuran File Biner     : 99 Byte
Compression Ratio (CR): 1.0101 : 1
Space Saving (SS)     : 1 %

[2] METRIK TEORETIS (TANPA METADATA / MURNI PAYLOAD)
------------------------------------------------
Original Size         : 800 bits
Compressed Size       : 394 bits
  - Pixel Values (AC) : 82 bits
  - Run Lengths       : 312 bits
Compression Ratio     : 49.25 %
Size Reduction        : 50.75 %
==========================================
```

Pada hasil untuk gambar yang relatif kecil dengan ukuran 10 x 10 piksel didapatkan ukuran berdasarkan ukuran nyata didapatkan berhasil melakukan kompresi lebih baik sebanyak 1% namun jika tidak dihitung dengan meta datanya secara teoritis algoritma ini dapat melakukan pemampatan sebesar 50.75% dengan mengunakan penggunaan pembagian lokalitas blok 8x8.

Dengan membandingkan pada makalah sebelumnya dengan gabungan RLE huffman Cr = 58.38% dan RLE fibonacci encoding Cr = 56.12% [6] tanpa menggunakan lokalitas 8x8.

```
--- C++ 1D Arithmetic Coding Analysis (Theoretical)
Original Size   : 800 bits
Compressed Size : 328 bits
  - Pixel Values: 80 bits
  - Run Lengths : 248 bits
Compression Ratio: 40.9249 %
Size reduction: 59.0751 %
------------------------------------------------
```

Pada citra berukuran kecil, secara teoretis metode berbasis blok 8 × 8 yang diusulkan menghasilkan rasio kompresi sebesar 49.25% (Kebutuhan bit terpangkas menjadi 394 bit). Nilai ini terbukti lebih baik dibandingkan dengan metode gabungan RLE-Huffman (58.38%) dan RLE-Fibonacci (56.12%) dari penelitian terdahulu [6] yang memproses citra secara linear tanpa lokalitas blok. Namun, terdapat penurunan performa yang signifikan ketika meninjau Metrik Dunia Nyata, di mana file biner fisik hanya mencatat penghematan ruang (Space Saving) sebesar 1% (99 Byte). Hal ini terjadi karena adanya beban fixed overhead dari Global Header (metadata dimensi gambar) yang ditulis oleh program C++. Pada citra berukuran sangat kecil, ukuran metadata ini menjadi hampir sebanding dengan ukuran data piksel itu sendiri. Selain itu, metode 1D Arithmetic Coding teoretis sedikit mengungguli metode 8x8 pada skala ini (40.92% vs 49.25%). Hal ini dikarenakan citra 10 ×10 piksel pada dasarnya sudah berukuran kecil, sehingga segmentasi blok menjadi rentan terhadap fragmentasi data lokal yang tidak perlu.

Berikutnya akan didilakukan analisis pada citra yang lebih besar berukuran 1177×1336 pixel

```
==========================================
            LAPORAN KOMPRESI CITRA 8x8
==========================================
```

```
[1] METRIK DUNIA NYATA (DENGAN METADATA / UKURAN FILE)
-------------------------------------------------
Ukuran Citra Asli     : 1572472 Byte
Ukuran File Biner     : 1283880 Byte
Compression Ratio (CR): 1.22478 : 1
Space Saving (SS)     : 18.3528 %

[2] METRIK TEORETIS (TANPA METADATA / MURNI PAYLOAD)
-------------------------------------------------
Original Size         : 1.25798e+07 bits
Compressed Size       : 1.10552e+07 bits
  - Pixel Values (AC) : 3.19869e+06 bits
  - Run Lengths       : 7.85647e+06 bits
Compression Ratio     : 87.8804 %
Size Reduction        : 12.1196 %
=================================================


- C++ 1D Arithmetic Coding Analysis (Theoretical)
-
Original Size   : 1.25798e+07 bits
Compressed Size : 1.44732e+07 bits
  - Pixel Values: 6.92978e+06 bits
  - Run Lengths : 7.54347e+06 bits
Compression Ratio: 115.052 %
Size reduction: -15.0517 %
-------------------------------------------------
```

Hasil dari analisis ini adalah dengan mengunakan prinsip lokalitas dan juga *arithmetic coding*. Dalam hasil ke dua ini mendapatkan anomali yaitu metrik teoritis mendapatkan nisbah yang lebih sedikit dibanding metrik dunia nyata hal ini bisa terjadi karena rumus yang dipakai estimasi ini malah melakukan estimasi berlebih pada bagian RLE karena masing-masing bagian RLE dalam implementasi program langsung dibebani dengan 8 bit sedangkan `BitstreamWriter` ini menuliskan data secara jauh lebih efisien melalui mekanisme tabel frekuensi terpadu per blok. Pada implementasi fisik, jika sebuah blok memiliki variasi simbol yang sedikit, `BitstreamWriter` hanya menyimpan intensitas piksel unik dan frekuensinya sekali saja di bagian header blok, sehingga terhindar dari penalti akumulasi 8-bit yang berulang kali terjadi pada setiap baris run-length di perhitungan simulasi teoretis..

V. KESIMPULAN DAN REKOMENDASI

Berdasarkan hasil perancangan, implementasi, serta pengujian yang telah dilakukan terhadap metode kompresi gabungan Run-Length Arithmetic Coding berbasis blok 8 × 8 dapat ditarik beberapa kesimpulan utama sebagai berikut

1. Strategi Decrease and Conquer Berhasil
   Pemecahan citra *grayscale* menjadi blok lokal 8 × 8 terbukti sukses mengisolasi komleksitas visual, shingga menurunkan nilai entropi lokal secara drastis.
2. Interval Probabilitas Optimal
   Lokalitas blok membangun pembagian interval probabilitas pada *artihmetic Coding* menjadi lebih lebar (nilai p tinggi). Hal ini mencegah penyusutan rentang interval yang ekstrem dan berhasil mengheat nilai pixsel hingga 53.8% dibanding metode 1D tanpa blok
3. Kompresi Fisik terbukti
   Integrasi metode ini pada citra grayscale besar menghasilkan performa nyata di penyimpanan komputer dengan nilai space saving space 18.35%
4. Unggil dari Metode Non-blok
   Pada citra kecil metode ini menghasilakn rasio komptesi teoritis terbesar 49.25% lebih efisien memampatkan data dibandingkan metode non-blok terdahulu seperti RLE huffman (58,38%) dan RLE-Fibonacci (56.12%)

Rekomendasi berikutnya adakan menggunakan metode *variable-length* coding untuk komponen RLE untuk menyimpan data panjang barisan piksel untuk meminimalkan penalti bit pada citra kompleks dan kembangkan algoritma ini untuk citra warna RGB.

REPOSITORI

https://github.com/HussainDzaki/Makalah-Stima.git

UCAPAN TERIMA KASIH

Saya bersyukur kepada Allah SWT untuk memberikan saya kesempatan dan ketahanan untuk menyelesaikan makalah ini tepat waktu. Saya juga berterima kasih kepada Bapak Dr. Ir. Rinaldi Munir, M.T. dan Ir. Rila Mandala sebagai dosen pengampu matakuliah Strategi Algoritma ini

REFERENSI

[1] Munir, Rinaldi. 2024. *Pengantar pemrosesan citra digital (bagian 1)* . https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/01-Pengantar-Pemrosesan-Citra-Digital-Bag1-2024.pdf (diakses pada 18 Juni 2026)

[2] Munir, Rinaldi. 2024. *Pembentukan Citra dan Digitalisasi Citra*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2022-2023/03-Pembentukan-Citra-dan-Digitalisasi-Citra-2022. (diakses pada 18 Juni 2026)

[3] Munir, Rinaldi. 2024. *Pemampatan Citra (bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Citra/2024-2025/25-Image-Compression-Bagian1-2024.pdf. (accessed on 18 Juni 2026)

[4] Munir, Rinaldi 2026. *Algoritma Decease and Conquer (bagian 1)*. https://informatika.stei.itb.ac.id/~rinaldi.munir/Stmik/2025-2026/11-Algoritma-Decrease-and-Conquer-2026-Bagian1.pdf (diakses pada 19 Juni 2026)

[5] Langdon, G. G. (1984). *An Introduction to Arithmetic Coding*. IBM Journal of Research and Development, 28(2), 135-149. https://www.cs.cmu.edu/~aarti/Class/10704/Intro_Arith_coding.pdf diakses pada 19 Juni 2026)

[6] Al Hussainy, Dzaki. 2025. Efficiency Comparison Between RLE-Fibonacci and RLE-Huffman Coding in Grayscale Image Compression. https://informatika.stei.itb.ac.id/~rinaldi.munir/Matdis/2024-2025-2/Makalah2025/Makalah-Matdis-2025-IF-ITB%20(87).pdf (diakses pada 19 juni 2026)

[7] G. K. Wallace, "The JPEG still picture compression standard," *IEEE Transactions on Consumer Electronics*, vol. 38, no. 1, pp. xviii-xxxiv, Feb. 1992, https://www.ijg.org/files/Wallace.JPEG.pdf (diakses pada 19 Juni 2026)

PERNYATAAN

Dengan ini saya menyatakan bahwa makalah yang saya tulis ini adalah tulisan saya sendiri, bukan saduran, atau terjemahan dari makalah orang lain, dan bukan plagiasi.

Bandung, 19 Juni 2026

Dzaki Ahmad Al Hussainy, 13524084