Pemampatan Citra Digital dengan Algoritma LZW

Muhammad Equilibrie Fajria - 13521047
Program Studi Teknik Informatika
Sekolah Teknik Elektro dan Informatika
Institut Teknologi Bandung, Jalan Ganesha 10 Bandung
E-mail (gmail): 13521047@std.stei.itb.ac.id

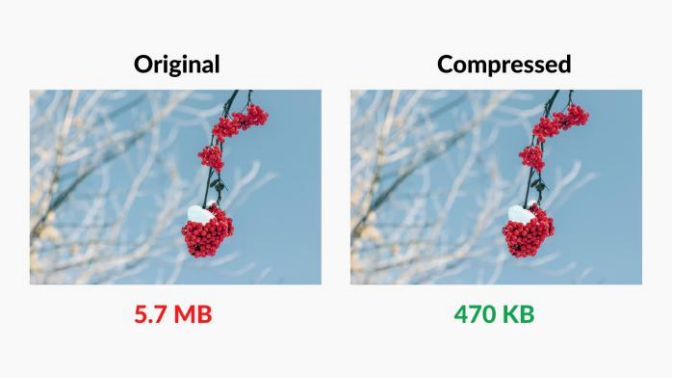
***Abstrak*—Dengan kemajuan teknologi yang pesat di era modern ini, kebutuhan akan penyimpanan dan transmisi data yang efisien semakin krusial, terutama untuk data visual seperti citra digital. Berhubung ukuran data visual yang biasanya relatif lebih besar dibandingkan dengan bentuk data lain, seperti teks dan audio, membuatnya seringkali menjadi tidak efisien dalam penyimpanan dan transmisi data. Salah satu metode untuk membuat data visual menjadi lebih efisien saat disimpan dan ditransmisikan adalah dengan mengurangi ukurannya menggunakan metode pemampatan. Metode pemampatan sangatlah beragam terbagi menjadi dua pendekatan utama, yakni pendekatan *lossy* dan pendekatan *lossless*. Pendekatan *lossy* memungkinkan hasil pemampatan menjadi sangat kecil, namun ada beberapa detail citra yang menjadi hilang. Detail ini biasanya tidak terasa perbedaannya di mata manusia. Sedangkan, pendekatan *lossless* memungkinkan hasil pemampatan untuk tetap menyimpan semua detail citra saat dinirmampatkan. Secara umum, pendekatan *lossless* dapat digunakan untuk semua kebutuhan. Salah satu algoritma yang menggunakan pendekatan *lossless* adalah algoritma *Lempel-Ziv-Welch* (LZW). Algoritma LZW menggunakan struktur data berupa kamus (*dictionary*) yang secara dinamis menyimpan kombinasi simbol atau karakter yang muncul selama proses pemampatan. Ketika algoritma menemukan urutan baru yang belum ada di kamus, maka urutan tersebut akan ditambahkan ke dalam kamus dengan indeks yang baru. Semakin banyak urutan yang berulang pada suatu citra, maka akan semakin efektif pula hasil pemampatan algoritma LZW. Atas alasan itulah, kinerja algoritma LZW sangat bergantung pada karakteristik citra. Citra dengan perulangan urutan piksel yang banyak dan variasi piksel yang sedikit akan menghasilkan pemampatan dengan ukuran kecil, begitu juga sebaliknya.**

Keywords—pemampatan; penirmampatan; citra digital; lzw; nisbah pemampatan; lossless, ukuran; bit

I. PENDAHULUAN

Pemampatan citra digital merupakan salah satu cabang yang memiliki peran yang sangat signifikan dalam bidang pemrosesan citra digital. Dalam era informasi yang semakin berkembang pesat, data dalam bentuk visual menjadi salah satu bentuk informasi yang paling sering digunakan untuk berbagai hal, seperti mengirim foto ke teman, mengunggah video pembelajaran ke situs web, dan menyimpan dokumentasi visual dari suatu acara. Pemampatan citra digital berfokus pada pengurangan ukuran data citra untuk memenuhi berbagai kebutuhan, seperti penyimpanan yang lebih efisien dan transmisi data yang lebih cepat. Peningkatan volume data yang dihasilkan oleh perangkat modern, seperti kamera digital beresolusi tinggi, ponsel pintar, drone, serta perangkat berbasis *Internet of Things* (IoT), menimbulkan tantangan tersendiri dalam pengelolaan dan distribusi data. Data visual ini tidak hanya memerlukan kapasitas penyimpanan yang besar, tetapi juga menghadapi berbagai kendala dalam pengiriman melalui jaringan dengan keterbatasan *bandwidth*.

Kondisi ini semakin diperparah oleh peningkatan permintaan akan kualitas citra yang tinggi dan kecepatan akses yang cepat. Misalnya, dalam industri media dan hiburan, konten video berkualitas tinggi seperti film dan siaran langsung memerlukan pemrosesan dan pengiriman data yang efisien agar dapat dinikmati oleh penonton dengan lancar tanpa adanya gangguan yang tidak diinginkan. Di sisi lain, aplikasi-aplikasi berbasis web dan mobile juga membutuhkan optimasi dalam pemampatan citra digital agar dapat memberikan pengalaman pengguna yang lebih baik. Dalam konteks ini, pemampatan citra digital tidak hanya sekadar menjadi solusi untuk mengurangi ukuran data, melainkan juga berperan penting untuk memastikan bahwa data dapat diakses dan ditransfer secara efisien.



Gambar 1. Ilustrasi Pemampatan Citra (Sumber: https://www.idownloadblog.com/2022/01/04/how-to-compress-image-mac/)

Berbagai metode pemampatan citra telah dikembangkan untuk memenuhi tuntutan ini, mulai dari teknik-teknik yang sederhana hingga algoritma yang lebih kompleks. Algoritma pemampatan ini dirancang untuk mengidentifikasi pola dan redundansi dalam data citra, sehingga dapat menghasilkan representasi yang lebih ringkas tanpa kehilangan informasi penting. Dengan demikian, pemampatan citra menjadi aspek fundamental dalam manajemen data visual, memungkinkan pengguna untuk mengoptimalkan penggunaan ruang

penyimpanan dan mempercepat proses transmisi data. Selain itu, perkembangan teknologi *cloud computing* juga ikut memengaruhi cara orang-orang menyimpan dan mengelola data. Pengguna mengharapkan akses yang cepat ke data mereka dari berbagai perangkat tanpa harus menghadapi kendala ukuran berkas yang besar.

Oleh karena itu, pengembangan algoritma pemampatan yang efektif dan efisien yang dapat diterapkan pada berbagai jenis data citra digital menjadi kebutuhan yang krusial dalam mendukung kemajuan teknologi masa kini. Salah satu algoritma yang dapat digunakan untuk melakukan pemampatan citra digital adalah algoritma *Lempel-Ziv-Welch* (LZW). Algoritma ini terkenal karena kemampuannya dalam melakukan kompresi *lossless*, yang mana prosesnya tidak menghilangkan data apapun dan data dapat dipulihkan seperti semula setelah proses pemampatan. Dalam makalah ini, akan dibahas lebih lanjut mengenai penerapan algoritma LZW untuk melakukan pemampatan citra digital. Hasil pemampatan citra digital menggunakan algoritma LZW akan diuji serta dianalisis lebih lanjut baik dari nisbah pemampatannya, kinerjanya, kelebihannya, hingga kekurangannya.

Dengan memahami pentingnya pemampatan citra digital dan teknologi di baliknya, diharapkan makalah ini dapat menyediakan wawasan bagi para peneliti dan praktisi di bidang teknologi informasi dan komunikasi. Temuan penelitian ini relevan tidak hanya untuk kebutuhan penyimpanan data tetapi juga dalam konteks aplikasi praktis, seperti kompresi citra untuk web, pengelolaan arsip digital, dan penghematan sumber daya dalam proses transmisi data. Selain itu, hasil penelitian ini dapat memberikan kontribusi terhadap pengembangan teknologi baru di bidang pemrosesan citra serta mendorong inovasi dalam aplikasi-aplikasi berbasis visual lainnya.

Penelitian ini juga membuka peluang untuk eksplorasi lebih lanjut mengenai algoritma-algoritma lain dalam konteks pemampatan citra serta dampaknya terhadap kualitas visual dan efisiensi penyimpanan di masa depan. Dengan meningkatnya kebutuhan akan solusi kompresi yang lebih baik seiring dengan pertumbuhan jumlah pengguna internet dan perangkat *mobile*, penting bagi para peneliti untuk terus mengeksplorasi metode baru serta meningkatkan algoritma yang sudah ada agar dapat memenuhi tuntutan tersebut.

II. LANDASAN TEORI

A. *Citra Digital*

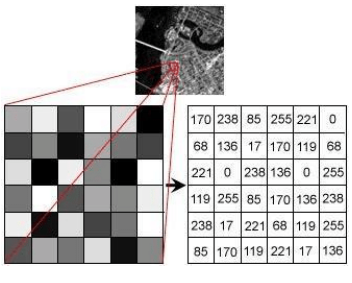
Citra dapat didefinisikan sebagai fungsi yang menggambarkan intensitas cahaya, dinyatakan dengan f(x,y) pada bidang dua dimensi (2D), dengan x dan y merupakan komponen dari sistem koordinat [2]. Nilai dari fungsi f pada titik koordinat (x,y) menunjukkan tingkat kecerahan citra di lokasi tersebut. Dalam konteks citra digital, baik nilai f maupun koordinat (x,y) bersifat diskrit, berbeda dengan citra kontinu yang memiliki nilai yang dapat bervariasi secara halus [4]. Oleh karena itu, citra digital dapat dipahami sebagai sebuah array besar yang terdiri dari titik-titik yang telah diambil sampelnya (*sampled*) dari citra kontinu. Setiap titik dalam array ini memiliki tingkat kecerahan yang terkuantisasi, yang berarti nilai kecerahan tersebut dibatasi pada level-level tertentu. Titik-titik ini dikenal sebagai piksel, yang merupakan elemen dasar penyusun citra digital.

Dalam pemrosesan citra digital, piksel tidak hanya menyimpan informasi mengenai kecerahan, tetapi juga warna [3]. Warna dalam citra digital umumnya direpresentasikan menggunakan model RGB (*Red*, *Green*, *Blue*), dengan setiap pikselnya terdiri dari tiga komponen warna yang masing-masing dapat memiliki nilai antara 0 hingga 255. Dengan kombinasi ketiga komponen ini, citra digital dapat merepresentasikan lebih dari 16 juta warna yang berbeda-beda.

Pemahaman mengenai struktur dan karakteristik piksel sangat krusial dalam analisis dan pemrosesan citra digital. Sebagai contoh, ketika mengembangkan algoritma untuk pemrosesan citra, penting bagi pengembang untuk mempertimbangkan interaksi antar piksel agar dapat menciptakan efek visual yang diinginkan. Selain itu, teknik pemampatan citra juga sangat bergantung pada pemahaman tentang bagaimana data piksel disimpan dan ditransmisi.

Citra digital memiliki berbagai macam tipe yang dapat digunakan. Setiap tipe citra digital memiliki teknik serta kebutuhan ukuran penyimpanan yang berbeda-beda. Terdapat setidaknya empat tipe dasar citra digital yang cukup sering digunakan, yaitu:

- Citra biner. Pada citra biner, setiap piksel hanya memiliki dua kemungkinan warna, yaitu hitam atau putih. Dengan begitu, ukuran penyimpanan yang diperlukan hanyalah 1-bit untuk setiap piksel. Hal ini membuat penyimpanan citra biner sangat efisien.
- Citra *grayscale*. Citra ini memiliki rentang warna abu-abu dengan nilai dari 0 hingga 255, dengan 0 mewakili warna hitam dan 255 mewakili putih. Setiap piksel pada citra grayscale biasanya direpresentasikan dengan 8-bit atau 1 byte.
- Citra *true color*/RGB. Piksel dalam citra RGB terdiri dari tiga warna dasar, yaitu merah, hijau, dan biru. Masing-masing warna tersebut memiliki rentang nilai antara 0 hingga 255. Untuk menyimpan setiap piksel dibutuhkan 24-bit atau 3 byte.
- Citra *indexed*. Pada citra indexed, setiap piksel menyimpan indeks yang merujuk ke color map atau palet warna yang berisi semua warna yang digunakan dalam citra tersebut. Pendekatan ini membuat penyimpanan citra indexed lebih efisien.

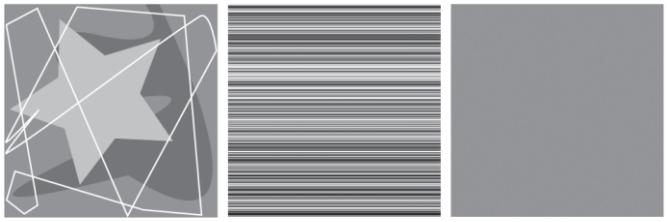


Gambar 2. Representasi Piksel Citra Digital (Sumber: https://www.researchgate.net/figure/Digital-image-representation-by-pixels-vii_fig2_311806469)

B. *Pemampatan Citra Digital*

Pemampatan data mengacu pada proses mengurangi jumlah data yang dibutuhkan untuk merepresentasikan sejumlah informasi tertentu [1]. Dalam definisi ini, data merupakan sarana untuk penyampaian informasi. Karena berbagai jumlah data dapat digunakan untuk merepresentasikan informasi yang sama, representasi yang berisi informasi tidak relevan atau berulang dapat dikatakan mengandung data yang redundan. Dalam konteks pemampatan citra digital, redundansi menjadi salah satu aspek utama yang dapat diidentifikasi dan dieksploitasi untuk mengurangi ukuran data tanpa kehilangan informasi penting. Terdapat tiga jenis redundansi utama yang dapat dieksploitasi dalam pemampatan citra digital, yaitu *coding redundancy*, *spatial and temporal redundancy*, serta *irrelevant information*.

Coding redundancy muncul ketika kode yang digunakan untuk merepresentasikan data memiliki panjang yang lebih besar daripada yang diperlukan. Misalnya, kode 8-bit yang digunakan untuk mewakili intensitas piksel sering kali mengandung lebih banyak bit daripada yang benar-benar diperlukan untuk merepresentasikan nilai intensitas tersebut. *Spatial and temporal redundancy* terjadi karena banyak piksel dalam citra atau bingkai video yang berkorelasi secara spasial atau temporal, sehingga informasi yang sama direplikasi berulang kali. Sementara itu, *irrelevant information* merujuk pada informasi yang tidak relevan bagi sistem visual manusia atau untuk tujuan tertentu, sehingga dapat diabaikan tanpa memengaruhi kualitas citra secara signifikan.



a b c

Gambar 3. Contoh redundansi: a) coding redundancy, b) spatial redundancy, dan c) irrelevant information [1]

Pemampatan citra digital bertujuan untuk mengurangi ukuran berkas citra dengan cara menghilangkan redundansi data tersebut, baik dengan mempertahankan semua informasi asli (*lossless compression*) maupun dengan menghapus informasi yang dianggap kurang penting (*lossy compression*). Proses ini sangat penting dalam pengelolaan data digital, terutama karena citra sering kali mengandung ukuran data yang besar, sehingga memerlukan kapasitas penyimpanan yang besar pula. Dengan melakukan pemampatan, baik untuk penyimpanan maupun transmisi, pengguna dapat menghemat ruang dan waktu, serta meningkatkan efisiensi dalam pengiriman data melalui jaringan yang memiliki keterbatasan *bandwidth* [2].

Dalam dunia pemampatan citra, terdapat dua pendekatan utama, yaitu *lossy* dan *lossless*. Pemampatan lossless sangat penting dalam aplikasi yang memerlukan data utuh, seperti pengolahan citra medis atau arsip digital. Algoritma seperti *Lempel-Ziv-Welch* (LZW), Huffman coding, dan DEFLATE merupakan beberapa teknik pemampatan *lossless* yang banyak digunakan.

Sebaliknya, pemampatan *lossy* sering kali digunakan dalam aplikasi yang penurunan kualitas citra dapat diterima, seperti fotografi digital, video *streaming*, atau multimedia online. Contohnya adalah algoritma JPEG, yang memanfaatkan transformasi Fourier dalam bentuk *Discrete Cosine Transform* (DCT) untuk menghilangkan detail frekuensi tinggi yang sulit terlihat oleh mata manusia [1]. Meskipun pemampatan *lossy* dapat mencapai rasio pemampatan yang lebih tinggi dibandingkan *lossless*, metode ini menghasilkan kehilangan informasi yang bersifat permanen, sehingga kualitas citra dapat menurun jika diperbesar atau dikompresi ulang.

Adapun metrik untuk melakukan pengukuran nisbah (rasio) pemampatan yang telah dilakukan pada suatu citra. Metrik tersebut dapat dilihat pada persamaan berikut [7]:

$$\text{Nisbah} = \frac{\text{Ukuran citra sesudah dimampatkan}}{\text{Ukuran citra sebelum dimampatkan}} \times 100\% \quad (1)$$

C. *Algoritma Lempel-Ziv-Welch* (LZW)

Algoritma *Lempel-Ziv-Welch* (LZW) merupakan teknik pemampatan data yang termasuk dalam kategori *lossless*, yang berarti data asli dapat dipulihkan sepenuhnya setelah proses pemampatan tanpa kehilangan informasi. Algoritma ini diciptakan oleh Abraham Lempel, Jacob Ziv, dan Terry Welch pada tahun 1977 dan telah banyak digunakan dalam berbagai aplikasi, termasuk format file seperti GIF, TIFF, dan PNG. LZW bekerja dengan mengatasi *coding redundancy* dan *spatial redundancy* pada data citra digital, sehingga menjadi salah satu metode pemampatan yang sangat efektif. Prinsip utama algoritma ini adalah menggunakan struktur data berupa kamus (*dictionary*) yang secara dinamis menyimpan kombinasi simbol atau karakter yang muncul selama proses pemampatan. Ketika algoritma menemukan urutan baru yang belum ada di kamus, maka urutan tersebut akan ditambahkan ke dalam kamus dengan indeks yang baru [2].

Proses kerja LZW dimulai dengan menginisialisasi kamus yang berisi semua simbol ASCII standar (256 karakter untuk data 8-bit). Algoritma membaca data secara berurutan, membangun string dari karakter yang dibaca. Jika string tersebut sudah ada dalam kamus, algoritma melanjutkan dengan menambahkan karakter berikutnya ke string tersebut. Namun, jika string baru tidak ada dalam kamus, algoritma menambahkannya sebagai entri baru dalam kamus dan menghasilkan indeks string sebelumnya sebagai output. Dengan cara ini, LZW dapat menggantikan urutan panjang data dengan referensi pendek ke dalam kamus, sehingga ukuran data yang dikompresi menjadi lebih kecil [5].

Sebagai contoh, dalam citra grayscale 8-bit, semua nilai intensitas piksel dari 0 hingga 255 dimasukkan ke dalam kamus pada awal proses. Ketika algoritma membaca sekuens intensitas seperti "255–255", urutan tersebut ditambahkan ke kamus di indeks berikutnya, misalnya 256. Jika urutan yang sama muncul lagi, algoritma cukup menggantinya dengan

indeks kamus (misalnya 256) alih-alih menyimpan urutan lengkap, sehingga redundansi data berkurang secara signifikan.

Currently Recognized Sequence	Pixel Being Processed	Encoded Output	Dictionary Location (Code Word)	Dictionary Entry
	39			
39	39	39	256	39–39
39	126	39	257	39–126
126	126	126	258	126–126
126	39	126	259	126–39
39	39			
39–39	126	256	260	39–39–126
126	126			
126–126	39	258	261	126–126–39
39	39			
39-39	126			
39–39–126	126	260	262	39–39–126–126
126	39			
126-39	39	259	263	126–39–39
39	126			
39-126	126	257	264	39–126–126
126		126		

Gambar 4. Contoh proses pemampatan menggunakan algoritma LZW [1]

Proses penirmampatan dalam algoritma LZW dimulai dengan menginisialisasi kamus yang sama seperti pada proses pemampatan, yaitu berisi 256 entri awal untuk semua nilai simbol dasar. *Decoder* lalu membaca setiap kode indeks dari data terkompresi secara berurutan dan menggunakan kamus untuk menerjemahkannya kembali ke string asli. Jika kode indeks telah ada di kamus, string yang sesuai langsung dikeluarkan. Ketika kode baru ditemukan, *decoder* memanfaatkan string sebelumnya yang telah diproses untuk merekonstruksi entri baru di kamus, yaitu dengan menambahkan karakter pertama dari string saat ini ke string sebelumnya. Jika kode baru belum ada di kamus (misalnya, saat kode dikompresi muncul sebelum entri kamus dibuat), *decoder* membuat entri baru dengan mengulangi karakter pertama dari string sebelumnya. Proses ini dilakukan secara iteratif hingga seluruh data terkompresi berhasil direkonstruksi menjadi data asli sepenuhnya, tanpa kehilangan informasi.

Keunggulan utama dari algoritma LZW adalah kemampuannya untuk beradaptasi dengan pola data yang berulang, baik pada teks maupun gambar. Rasio kompresi yang dicapai sangat bergantung pada struktur data yang dikompresi. Pada data yang memiliki banyak pola berulang, seperti citra digital atau teks terstruktur, algoritma ini dapat mencapai efisiensi yang sangat tinggi. Namun, ada beberapa kelemahan yang perlu diperhatikan. Jika data yang dikompresi memiliki banyak variasi atau tidak ada pola yang jelas, seperti pada citra dengan noise tinggi, nisbah pemampatan mungkin tidak akan seefektif pada data yang lebih terstruktur. Selain itu, ukuran kamus juga dapat menjadi kendala jika data yang dikompresi sangat besar, karena dapat meningkatkan kompleksitas pemrosesan [6].

III. Implementasi Solusi

Terdapat 3 bagian utama program yang diimplementasikan, yaitu bagian algoritma pemampatan LZW, algoritma penirmampatan LZW, dan bagian antarmuka (GUI) yang berfungsi menampilkan dan menyediakan fungsi-fungsi pemampatan/penirmampatan kepada pengguna. Solusi diimplementasikan menggunakan bahasa Python dengan *library* PIL untuk mengelola citra dan tkinter untuk mengelola antarmuka (GUI).

A. Algoritma Pemampatan LZW

Berikut adalah kode program untuk algoritma pemampatan LZW.

```
# LZW compresssion function
def compress(data, max_bits):
    dictionary = {chr(i): i for i in range(256)}
    next_code = 256
    result = []
    w = ""

    max_size = 2 ** max_bits

    for c in data:
        wc = w + c
        if wc in dictionary:
            w = wc
        else:
            result.append(dictionary[w])
            if next_code < max_size:
                dictionary[wc] = next_code
                next_code += 1
            w = c

    if w:
        result.append(dictionary[w])

    return result
```

Algoritma pemampatan *Lempel-Ziv-Welch* (LZW) pada kode program di atas bekerja dengan cara mengganti urutan karakter dalam data dengan kode numerik yang lebih pendek menggunakan sebuah kamus (*dictionary*). Proses pemampatan dimulai dengan membuat kamus yang berisi karakter-karakter ASCII standar (dari 0 hingga 255) dan memberi kode numerik yang sesuai. Setiap karakter dalam data kemudian diproses satu per satu. Untuk setiap karakter, algoritma menggabungkan karakter tersebut dengan substring sebelumnya (w + c). Jika substring tersebut sudah ada di kamus, algoritma memperbarui substring yang sedang diproses (w) menjadi substring baru ini. Jika tidak, program menambahkan kode numerik untuk substring sebelumnya ke dalam hasil kompresi, kemudian menambahkan substring baru tersebut ke dalam kamus dan melanjutkan pemrosesan dengan substring baru. Setelah seluruh data diproses, algoritma memastikan bahwa substring terakhir juga dimasukkan ke dalam hasil kompresi. Pada algoritma terdapat mekanisme

untuk membatasi jumlah bit maksimum (max_bits) yang digunakan dalam proses pemampatan dan penirmampatan. Pembatasan ini berguna untuk menghindari penambahan ukuran kamus yang tidak terkendali dan memastikan pemampatan tetap efisien.

B. Algoritma Penirmampatan LZW

Berikut adalah kode program untuk algoritma penirmampatan LZW.

```
# LZW decompresssion function
def decompress(compressed, max_bits):
    dictionary = {i: chr(i) for i in range(256)}
    next_code = 256
    result = ""
    w = chr(compressed.pop(0))
    result += w

    max_size = 2 ** max_bits

    for k in compressed:
        if k in dictionary:
            entry = dictionary[k]
        elif k == next_code:
            entry = w + w[0]
        else:
            raise ValueError("Kompresi rusak atau tidak valid.")

        result += entry

        if next_code < max_size:
            dictionary[next_code] = w + entry[0]
            next_code += 1
        w = entry

    return result
```

Pada bagian penirmampatan, proses dimulai dengan membuat kamus yang sama seperti bagian awal proses pemampatan, yaitu karakter-karakter ASCII standar (dari 0 hingga 255) dengan kode numeriknya. Setiap kode numerik dalam data yang terkompresi dicocokkan dengan entri yang ada di kamus. Jika kode numerik ada di kamus, substring yang sesuai akan ditambahkan ke hasil dekompresi. Jika kode numerik tersebut merupakan kode baru (yaitu kode yang belum ada di kamus), maka entri baru akan dibuat dengan cara menggabungkan karakter pertama dari substring sebelumnya (w + w[0]). Selanjutnya, kode baru ini ditambahkan ke kamus untuk digunakan dalam pemrosesan selanjutnya. Proses ini terus berlanjut hingga seluruh data selesai dinirmampatkan sepenuhnya.

C. Antarmuka (GUI)

Untuk bagian antarmuka, digunakan beberapa fungsi pemrosesan citra. Beberapa fungsi yang relevan dapat dilihat pada cuplikan kode program berikut.

```
def compress_image(self):
    # Open image and get pixel data
    img = Image.open(self.image_path)
    mode = img.mode
    if mode != "L":
        img = img.convert("RGB")
    pixels = list(img.getdata())

    if mode == "L":
        # For grayscale, one channel
        pixel_data = "".join(chr(pixel) for pixel in pixels)
    else:
        # For RGB, three channels
        pixel_data = "".join(chr(value) for pixel in pixels for value in pixel)

    compressed = LZW.compress(pixel_data, self.max_bits)
    self.compressed_data = (compressed, mode, img.size) # Simpan mode dan ukuran

    original_size = len(pixel_data) * 8

    # Count compressed size
    current_bits = 8
    compressed_size = 0
    for code in compressed:
        if code >= (1 << current_bits):
            current_bits += 1
        compressed_size += current_bits

    compression_ratio = compressed_size / original_size * 100

def upload_compressed(self):
    decompressed_data = LZW.decompress(compressed_data, self.max_bits)

    if mode == "L":
        # For grayscale
        pixels = [ord(char) for char in decompressed_data]
    else:
        # For RGB
        pixels = [
            tuple(ord(decompressed_data[i + j]) for j in range(3))
            for i in range(0, len(decompressed_data), 3)
        ]

    # Create image from pixel data
    img = Image.new(mode, size)
    img.putdata(pixels)
```

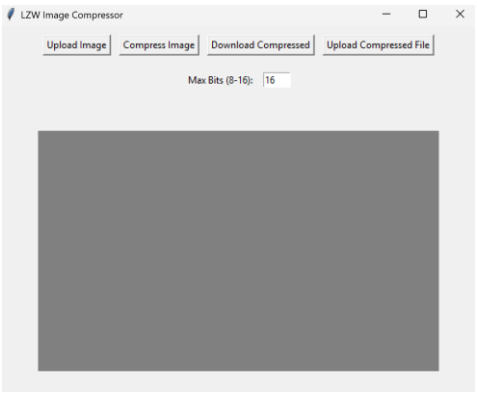
Fungsi compress_image akan membuka citra berdasarkan masukan dari pengguna, lalu memeriksa tipe citra. Pada program ini, diasumsikan pengguna hanya akan memasukkan citra bertipe *grayscale* dan RGB. Jika citra bertipe RGB, maka citra akan diratakan menjadi array dengan elemen tunggal dari yang awalnya berelemen tupple. Kemudian fungsi ini akan

memanggil fungsi compress dari kode LZW untuk memampatkan piksel-piksel citra masukan. Setelah selesai memampatkan, akan dihitung ukuran citra asli dan citra hasil pemampatan dalam satuan bit. Citra asli diasumsikan selalu menggunakan 8-bit untuk menyimpan piksel. Untuk citra hasil pemampatan, perhitungan dilakukan dengan menghitung seluruh kode numerik hasil pemampatan mulai dari 8-bit, bertahap bertambah 1-bit sesuai dengan kode numeriknya. Kemudian, dihitung nisbah pemampatan sesuai dengan persamaan (1).

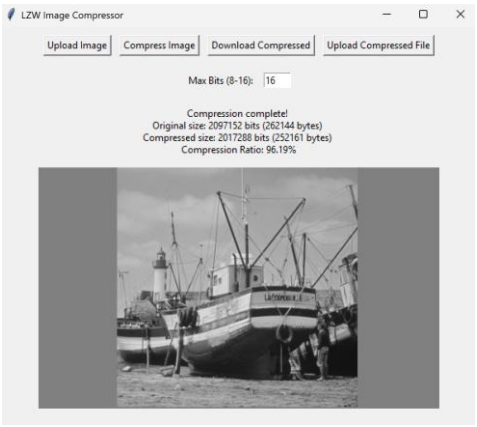
Untuk fungsi upload_compressed, setelah citra yang dimampatkan dimasukkan oleh pengguna, dilakukan penirmampatan. Penirmampatan dilakukan dengan memanggil fungsi decompress dari kode LZW. Hasil penirmampatan lalu dipetakan sesuai dengan tipe awalnya sehingga terbentuk citra semula yang *lossless*.

IV. Hasil dan Analisis

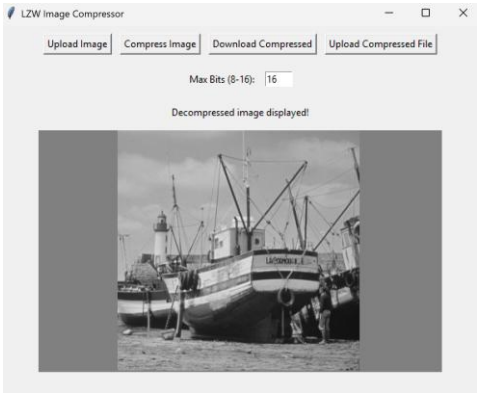
Pada bab ini, akan dilakukan pengujian program beserta analisis menggunakan beberapa citra grayscale dan berwarna. Berikut adalah tangkapan layar antarmuka (GUI) dari program.



Gambar 5. Tampilan Awal Antarmuka



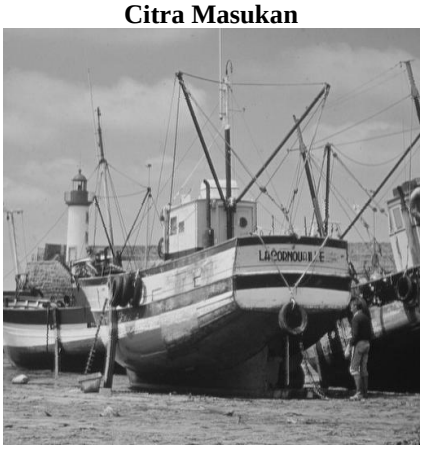
Gambar 6. Tampilan Pemampatan Antarmuka



Gambar 7. Tampilan Penirmampatan Antarmuka

Berikut adalah tangkapan layar dari hasil pengujian dengan citra uji.

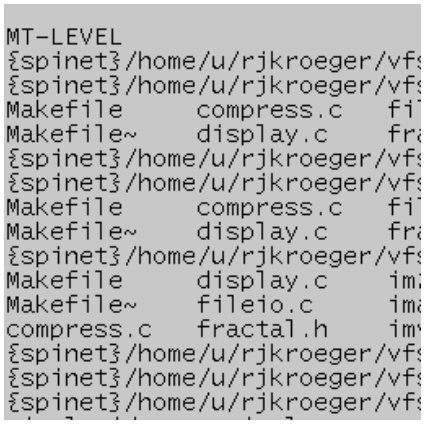
Citra Masukan



Ukuran asli: 2.097.152 bits

Max Bits: 10	**Max Bits: 12**	**Max Bits: 14**	**Max Bits: 16**
Ukuran terkompresi: 2.141.723 bits Nisbah pemampatan: 102,13%	Ukuran terkompresi: 2.286.713 bits Nisbah pemampatan: 109,04%	Ukuran terkompresi: 2.289.740 bits Nisbah pemampatan: 109.18%	Ukuran terkompresi: 2.017.288 bits Nisbah pemampatan: 96,19%

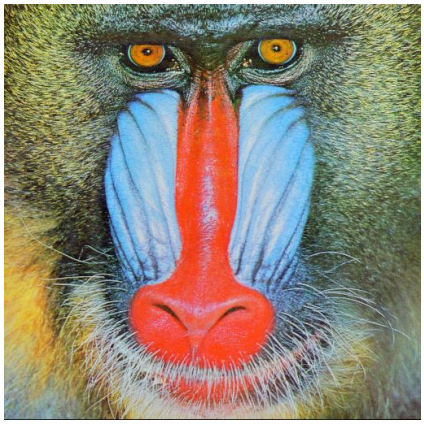
Citra Masukan



Ukuran asli: 524.288 bits

Max Bits: 10	**Max Bits: 12**	**Max Bits: 14**	**Max Bits: 16**
Ukuran terkompresi: 35.118 bits Nisbah pemampatan: 6,7 %	Ukuran terkompresi: 34.413 bits Nisbah pemampatan: 6,56 %	Ukuran terkompresi: 34.413 bits Nisbah pemampatan: 6.56 %	Ukuran terkompresi: 34.413 bits Nisbah pemampatan: 6,56 %

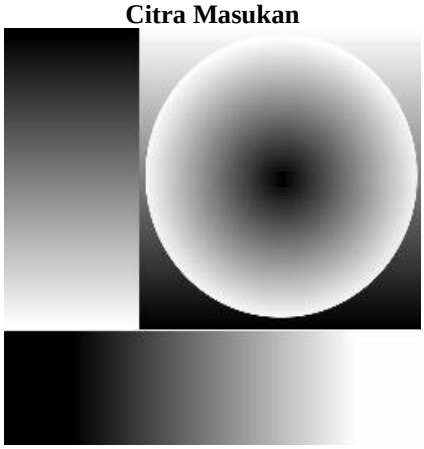
Citra Masukan



Ukuran asli: 6.291.456 bits

Max Bits: 10	**Max Bits: 12**	**Max Bits: 14**	**Max Bits: 16**
Ukuran terkompresi: 7.652.410 bits Nisbah pemampatan: 121,63 %	Ukuran terkompresi: 8.350.294 bits Nisbah pemampatan: 132,72 %	Ukuran terkompresi: 7.962.971 bits Nisbah pemampatan: 126,57 %	Ukuran terkompresi: 7.470.155 bits Nisbah pemampatan: 118,73 %

Citra Masukan



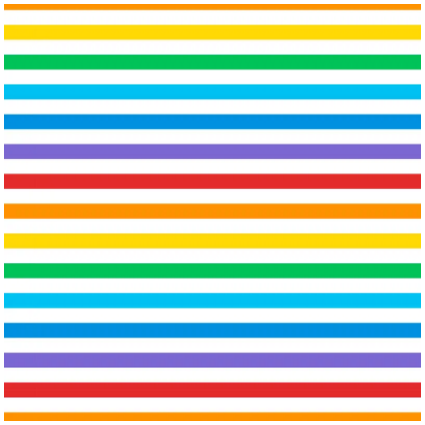
Ukuran asli: 524.288 bits

Max Bits: 10	**Max Bits: 12**	**Max Bits: 14**	**Max Bits: 16**
Ukuran terkompresi: 510.122 bits Nisbah pemampatan: 97,3 %	Ukuran terkompresi: 383.598 bits Nisbah pemampatan: 73,17 %	Ukuran terkompresi: 210.460 bits Nisbah pemampatan: 40,14 %	Ukuran terkompresi: 210.460 bits Nisbah pemampatan: 40,14 %

Citra Masukan

Ukuran asli: 1.572.864 bits

Max Bits: 10	**Max Bits: 12**	**Max Bits: 14**	**Max Bits: 16**
Ukuran terkompresi: 1.236.953 bits Nisbah pemampatan: 78,64 %	Ukuran terkompresi: 1.147.888 bits Nisbah pemampatan: 72,98 %	Ukuran terkompresi: 936.679 bits Nisbah pemampatan: 59,55 %	Ukuran terkompresi: 585.511 bits Nisbah pemampatan: 37,23 %

Citra Masukan			
			
Ukuran asli: 15.360.000 bits			
Max Bits: 10 Ukuran terkompresi: 18.670.198 bits Nisbah pemampatan: 121,55 %	**Max Bits: 12** Ukuran terkompresi: 12.696.905 bits Nisbah pemampatan: 82,66 %	**Max Bits: 14** Ukuran terkompresi: 9.748.384 bits Nisbah pemampatan: 63,47 %	**Max Bits: 16** Ukuran terkompresi: 1.459.623 bits Nisbah pemampatan: 9,5 %

Berdasarkan hasil pengujian, dapat dilihat bahwa secara umum penambahan bit maksimal yang dapat digunakan dalam pemampatan LZW akan mengurangi ukuran hasil pemampatan dan nisbah pemampatan. Hal ini berlaku terutama untuk citra yang memiliki pola pengulangan piksel sama yang banyak dengan variasi pixel yang kurang beragam. Hal ini juga membuat pemampatan ini lebih efektif untuk citra *grayscale* dibandingkan dengan citra RGB.

Namun, untuk beberapa citra dengan variasi piksel yang beragam, pemampatan LZW dapat menambah ukuran citra. Hal ini dikarenakan citra yang awalnya hanya membutuhkan 8-bit untuk menyimpan setiap elemen piksel citra menjadi membutuhkan lebih banyak bit. Hal ini membuat pemampatan LZW tidak sesuai digunakan untuk citra dengan variasi piksel yang beragam.

V. Kesimpulan dan Saran

Setelah berhasil mengimplementasikan program pemampatan LZW, dapat dilihat bahwa program pemampatan LZW berfungsi dengan baik. Ukuran hasil pemampatan LZW sangat bergantung pada karakteristik citra yang akan dimampatkan. Dengan demikian, penggunaan algoritma LZW untuk melakukan pemampatan tidak dapat secara sembarangan karena memiliki potensi menambah ukuran citra jika tidak diimplementasikan dan digunakan secara tepat. Selain itu, untuk penggunaan kasus nyata diperlukan juga manipulasi bit yang efisien dan efektif untuk menyimpan hasil pemampatan pada sebuah *file*. Bagian ini tidak menjadi bahasan pada makalah ini dikarenakan kompleksitasnya yang cukup tinggi.

Secara umum, masih banyak ruang untuk pengembangan dalam implementasi algoritma LZW, terutama untuk bagian penyimpanan pada *file*. Terdapat potensi untuk menggabungkan algoritma LZW dengan algoritma pemampatan lain, seperti RLE dan Huffman untuk dapat mencapai nisbah pemampatan yang lebih baik. Perlu dilakukan riset dan percobaan lebih lanjut untuk mengembangkan kinerja dari pemampatan menggunakan algoritma LZW.

Pranala Video Youtube

https://youtu.be/74_pOztIUns

Pranala Github

https://github.com/MuhLibri/LZW-Image-Compressor

Ucapan Terima Kasih

Puji dan syukur penulis panjatkan kepada Tuhan Yang Maha Esa, atas berkat dan rahmat-Nya makalah yang berjudul "Pemampatan Citra Digital dengan Algoritma LZW" dapat terselesaikan. Tidak lupa penulis juga mengucapkan terima kasih kepada dosen mata kuliah Pemrosesan Citra Digital, Bapak Dr. Ir. Rinaldi, M.T., yang telah mengajar dan membimbing penulis pada mata kuliah Pemrosesan Citra Digital selama satu semester. Penulis juga ingin mengucapkan terima kasih kepada teman-teman dan keluarga penulis yang telah memberikan dukungan kepada penulis. Terakhir, penulis juga ingin meminta maaf apabila terdapat kesalahan dalam pembuatan makalah ini.

Referensi

[1] Gonzalez, R. C. & Woods, R. E., Digital Image Processing Fourth Edition, Pearson, 2009.

[2] Kok, C. W. & Tam, W. S., Digital Image Interpolation in MATLAB First Edition, John Wiley & Sons Singapore Pte. Ltd., 2019.

[3] Mangaras Y. F., Bambang Y., & Dessyanto B. P., Dasar Pengolahan CItra Digital Edisi 2022, Yogyakarta: Lembaga Penelitian dan Pengabdian kepada Masyarakat UPN Veteran Yogyakarta, 2022.

[4] A. McAndrew, A Computational Introduction to Digital Image Processing (2nd ed.), Melbourne: CRC Press, 2016.

[5] M. Dipperstein, "Lempel-Ziv-Welch (LZW) Encoding Discussion and Implementation.," 1998.

[6] Alim, A.N., Yuana, H., & Febrinita, F., "Aplikasi Kompresi Citra dengan Menggunakan Algoritma Lempel Ziv Welch (LZW)," *Jurnal Mahasiswa Teknik Informatika*, vol. 6, no. 2, pp. 684-695, 2022.

[7] R. Munir, Pemampatan Citra Bagian 2, Bandung: Program Studi Teknik Informatika, 2024.

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, 15 Januari 2025

Muhammad Equilibrie Fajria - 13521047