KINERJA DAN PERFORMA ALGORITMA KOMPRESSI LOSSLESS TERHADAP OBJEK CITRA DIGITAL

Aditya Wijaya, Suryarini Widodo

Jurusan Teknik Informatika Fakultas Teknologi Industri Univesitas Gunadarma
Jl. Margonda Raya No. 100 Pondok Cina Depok INDONESIA 16424
Telp. 021-78881112

adityawijaya@student.gunadarma.ac.id, srini@staff.gunadarma.ac.id

Abstrak

*Kompresi pada citra digital berfungsi untuk memampatkan data agar dapat menghemat tempat penyimpanan dengan atau tanpa mengurangi kualitas dari citra. Dalam teknik pengolahan citra untuk mendapatkan citra dengan kualitas tinggi dibutuhkan memori penyimpanan yang tidak sedikit sehingga kebutuhan akan penggunaan memori penyimpanan dirasakan sangat tinggi. Oleh karena itu diterapkanlah metode kompresi terhadap berbagai objek data guna mengurangi beban memori yang dimiliki, dalam hal ini adalah citra digital, tanpa harus mengurangi informasi data tersebut saat dilakukan proses pengembalian data atau dekompresi. Ini berarti metode yang digunakan bersifat lossless. Paper ini mencoba menganalisa untuk mengetahui kinerja dan performa dari beberapa algoritma kompresi untuk objek citra digital yang terbagi menjadi dua jenis yaitu citra berwarna dan grayscale dengan format *.jpg, *.bmp, *.tif, Algoritma yang akan diuji adalah Algoritma Huffman, LZW dan RLE yang merupakan metode kompresi lossless. Setiap algoritma kompresi memiliki kompleksitas yang berbeda – beda, baik dari segi kinerja pengkompresian ataupun performa dalam hal ini kecepatan melakukan kompresi dan dekompresi. Algoritma Huffman secara keseluruhan dapat di nilai baik karena memiliki kompabilitas tinggi untuk dapat mengkompresi format objek citra. Algoritma LZW menghasilkan performa waktu kompresi dan dekompresi yang kurang baik tetapi algoritma ini memiliki kinerja kompresi yang tinggi. Algoritma RLE, memiliki kompabilitas kemampuan yang tidak cukup bagus dari kedua algoritma sebelumnya. Untuk menguji algoritma kompresi lossless pada citra digital ini, dibuat modul program menggunakan bahasa C++, guna menguji objek citra digital.*

Kata kunci: Kompresi, Dekompresi, Lossless, Citra Digital

1 Pendahuluan

Kompresi adalah proses mengubah data menjadi sekumpulan kode untuk menghemat tempat penyimpanan dengan atau tanpa mengurangi kualitas dari citra serta mempercepat waktu transmisi data. Pada penggunaannya, ada beberapa faktor yang sering menjadi pertimbangan dalam memilih suatu metode kompresi, yaitu kecepatan kompresi, sumber daya yang dibutuhkan (memori, kecepatan PC), ukuran file hasil kompresi serta kompleksitas algoritma, karena pada intinya tidak setiap metode kompesi sesuai untuk semua jenis file gambar.

Tujuan dari kompresi terhadap citra digital ini adalah untuk mengurangi redudansi dari data - data yang terdapat dalam citra sehingga dapat disimpan atau ditransmisikan secara efisien. Masalah inilah yang menyebabkan munculnya kebutuhan akan kompresi citra tanpa harus mengurangi informasi yang tersimpan didalam citra tersebut (*lossless*).

Paper ini mencoba untuk menganalisa beberapa algoritma kompresi lossless yaitu algoritma Huffman, Lemple-Ziv-Welch (LZW) dan Run Length Encoding (RLE) untuk mengetahui performa dan kinerja kompresi dan dekompresi dari algoritma tersebut dan juga untuk menngetahui ratio kompresi perbandingan memori citra sebelum dan setelah dilakukan

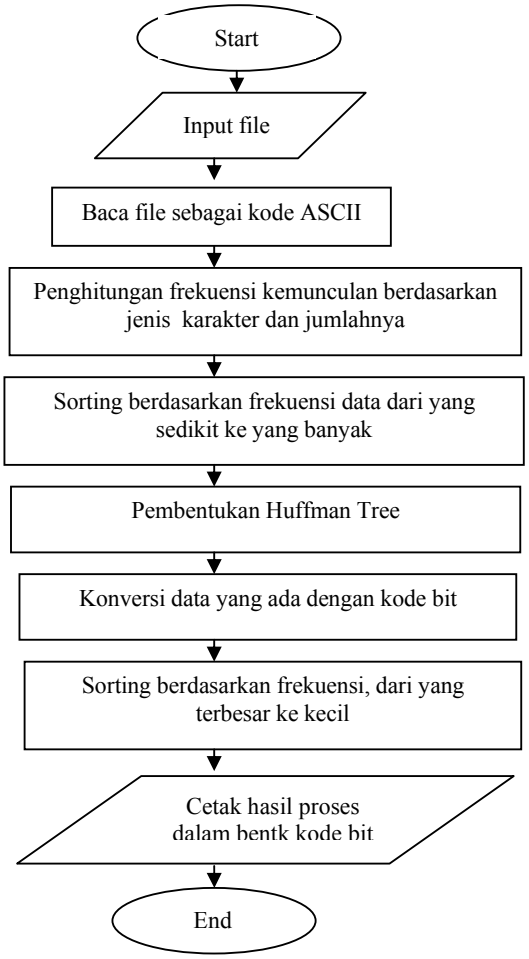
2 Kompresi Lossless

Berdasarkan fungsinya, metode kompresi dibagi menjadi dua yaitu metode *lossy* dan metode *lossless*. Metode Lossless adalah suatu metode pemampatan data tanpa menghilangkan data yang dimilikinya. Proses kompresi dapat dilakukan dengan mengganti suatu data dengan kode yang berukuran lebih ringkas. Data yang telah diganti dapat dikembalikan dengan ukuran dan struktur yang sama persis dengan data asli melalui proses dekompresi. Teknik ini bersifat dua arah, karena untuk membaca data yang telah terkompres diperlukan proses dekompresi. Berbeda dengan algoritma lossy, file hasil kompresi dengan metode lossless tak dapat dibaca tanpa melalui proses dekompresi. Teknik dekompresi ini bersifat spesial untuk satu teknik kompresi. Sehingga tidak mungkin file yang terkompres oleh oleh algoritma kompresi A dapat di dekompres dengan algoritma dekompres B. Tujuan utama dari metode lossless adalah menghasilkan file yang kecil namun tanpa sedikitpun mengurangi kualitas data tersebut. Hal ini sangat penting untuk data-data yang memerlukan keakuratan yang tinggi. Contoh teknik kompresi lossless diantaranya adalah Huffman dan Lempel-Ziv-Welch, Dynamic Markov Compression.

Algoritma Huffman merupakan salah satu metode yang paling lama dan paling terkenal dalam kompresi teks. Algoritma Huffman menggunakan prinsip pengkodean yang mirip dengan kode Morse, yaitu tiap karakter (simbol) dikodekan hanya dengan rangkaian beberapa bit, dimana karakter yang sering muncul dikodekan dengan rangkaian bit yang pendek dan karakter yang jarang muncul dikodekan dengan rangkaian bit yang lebih panjang.

Tahapan proses kompresi algoritma Huffman :

a. Hitung banyaknya jenis karakter dan jumlah dari masing-masing karakter yang terdapat dalam sebuah file.
b. Susun setiap jenis karakter dengan urutan jenis karakter yang jumlahnya paling sedikit ke yang jumlahnya paling banyak.
c. Buat pohon biner berdasarkan urutan karakter dari yang jumlahnya terkecil ke yang terbesar, dan memberi kode untuk tiap karakter.
d. Ganti data yang ada dengan kode bit berdasarkan pohon biner.
e. Simpan jumlah bit untuk kode bit yang terbesar, jenis karakter yang diurutkan dari frekuensi keluarnya terbesar ke terkecil beserta data yang sudah berubah menjadi kode bit sebagai data hasil kompresi.

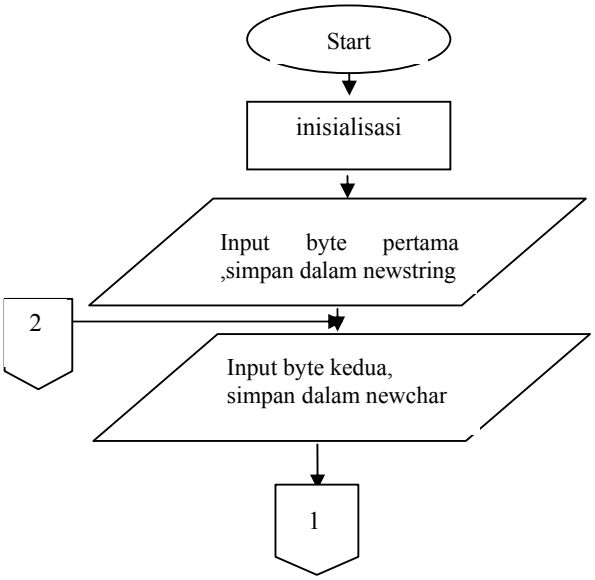


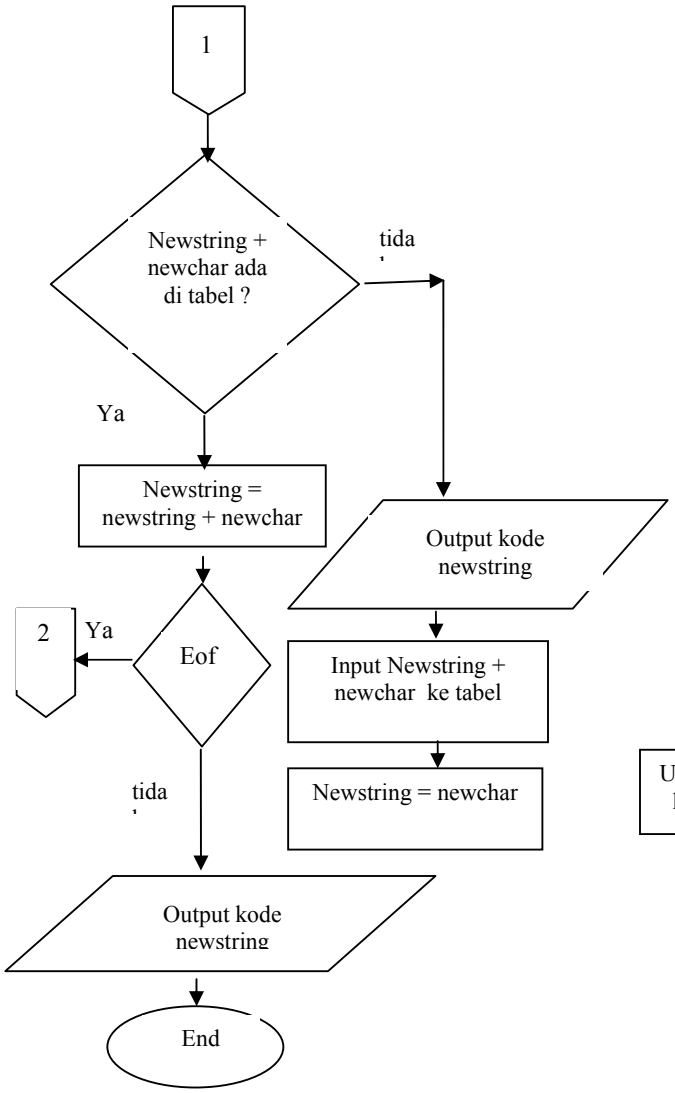
Gambar 1. Flowchart algoritma Huffman

Lempel-Ziv-Welch (LZW) adalah algoritma kompresi lossless yang dirancang untuk cepat dalam implementasi teapi biasanya tidak optimal karena hanya melakukan analisis terbatas pada data. Algoritma ini melakukan kompresi dengan menggunakan dictionary, dimana fragmen-fragmen teks digantikan dengan indeks yang diperoleh dari sebuah "kamus". Prinsip sejenis juga digunakan dalam kode Braille, di mana kode-kode khusus digunakan untuk merepresentasikan kata-kata yang ada. Pendekatan algoritma ini bersifat adaptif dan efektif karena banyak karakter dapat dikodekan dengan mengacu pada string yang telah muncul sebelumnya dalam teks. Prinsip kompresi tercapai jika referensi dalam bentuk pointer dapat disimpan dalam jumlah bit yang lebih sedikit dibandingkan string aslinya.

Tahapan proses dalam algoritma LZW :

1. Dictionary diinisialisasi dengan semua karakter dasar yang ada {'A'..'Z','a'..'z','0'..'9'}.
2. a = input pertama dalam *stream* newstring.
3. x = input berikutnya dalam *stream* newchar.
4. Apakah *string* ($a + x$) terdapat dalam *dictionary* ?
   a. Jika ya, maka $a = a + x$ (gabungkan a dan x menjadi *string* baru).
   b. Jika tidak, maka ;
      i. *Output* sebuah kode untuk menggantikan *string* a.
      ii. Tambahkan *string* ($a + x$) ke dalam *dictionary* dan berikan nomor/kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut.
      iii. $a = x$
5. Apakah masih ada karakter berikutnya dalam *stream* karakter ?
   a. Jika ya, maka kembali ke langkah 2.
   b. Jika tidak, maka *output* kode yang menggantikan *string a*, lalu terminasi proses (*stop*).



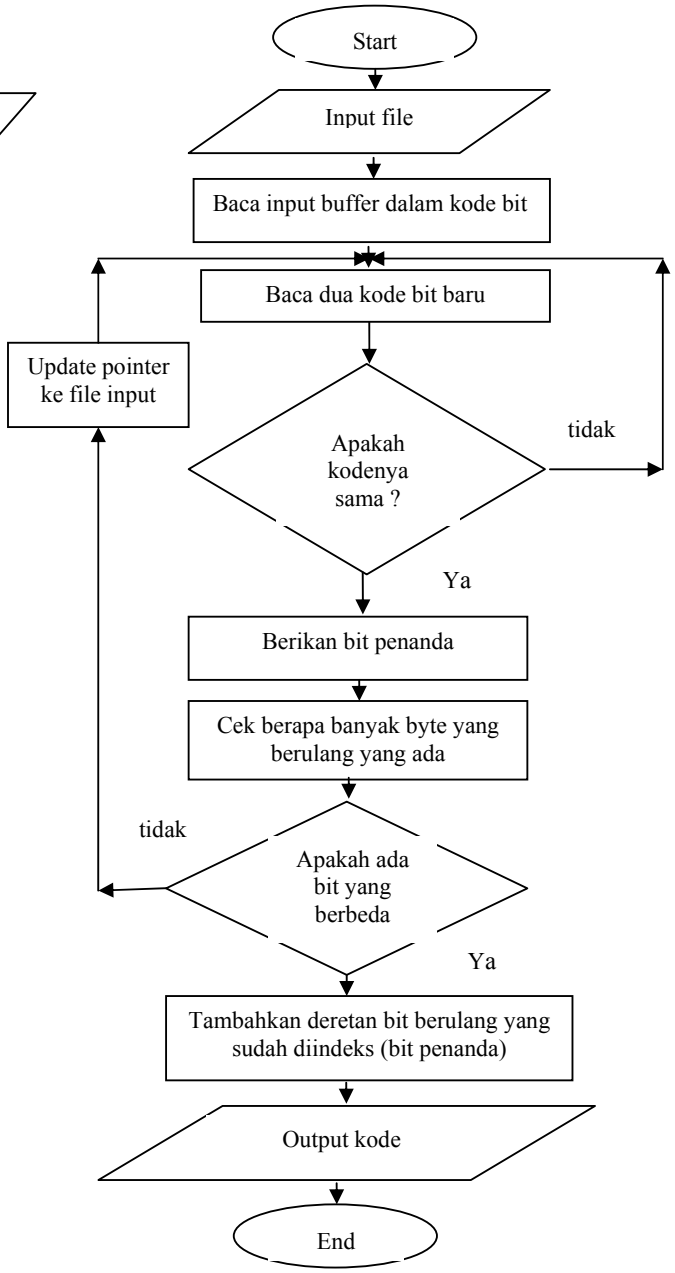


Gambar 2. Flowchart algoritma LZW

Algoritma Run-length digunakan untuk memampatkan data yang berisi karakter – karakter berulang. Saat karakter yang sama diterima secara berderet lebih dari tiga, algoritma ini mengkompres data dalam suatu tiga karakter berderetan. Algoritma Run-length paling efektif pada file – file grafis, dimana biasanya berisi deretan karakter yang sama.

Langkah didalam proses algoritma *Run-Length* :

1. Lihat apakah terdapat deretan karakter yang sama secara berurutan lebih dari tiga karakter, jika memenuhi lakukan pemampatan. Misal pada deretan karakter yang sama secara berurutan sebanyak 8 karakter, jadi lebih dari tiga dan dapat dilakukan pemampatan.
2. Berikan bit penanda pada file pemampatan, bit penanda disini berupa 8 deretan bit yang boleh dipilih sembarang asalkan digunakan secara konsisten pada seluruh bit penanda pemampatan. Bit penanda ini berfungsi untuk menandai bahwa karakter selanjutnya adalah karakter pemampatan sehingga tidak membingungkan pada saat mengembalikan file yang sudah dimampatkan ke file aslinya. Misal bit penanda ini dipilih 11111110.
3. Tambahkan deretan bit untuk menyatakan jumlah karakter yang sama berurutan, pada contoh diatas karakter yang sama berturutan sebanyak delapan kali, jadi diberikan deretan bit 00001000 (8 desimal).
4. Tambahkan deretan bit yang menyatakan karakter yang berulang, misal pada contoh diatas karakter yang berulang adalah 01000001 atau karakter A pada karakter ASCII.



Gambar 1 Flowchart algoritma RLE

3 Pengujian Kompresi Huffman, LZW dan RLE

Pengujian dilakukan terhadap format citra bmp, jpg, dan tiff. Dimana pengujian akan dilakukan dengan tiga objek citra untuk tiap formatnya, baik dalam citra berwarna ataupun grayscale serta dengan resolusi yang berbeda – beda. Hal ini untuk mengetahui kemampuan kompresi dari ketiga.

Gambar 4. Objek citra berwarna yang diuji
(a. Motor.bmp, b Heli.jpg, dan c. Mobil.tiff)

Gambar5. Objek citra grayscale yang diuji
(a. Ubuntu.bmp, b. Sungai.jpg, dan c. Night.tiff)

Pada pengujian pertama disini pada objek citra bewarna, lebih jelasnya dapat dilihat pada tabel 1 yang menerangkan tentang hasil dari pengujian terhadap citra berwarna.

Tabel 1. Citra berwarna setelah di uji modul Huffman

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
motor.bmp	1600x1200	5,760,054	5,352,049	7.09%	109 ms	531 ms
city.bmp	700x525	1,102,554	1,084,149	1.67%	16 ms	110 ms
ball.bmp	950x713	2,033,530	1,721,216	15.36%	47 ms	171 ms
heli.jpg	1024x768	287,566	285,635	0.68%	0 ms	31 ms
bear.jpg	1200x900	693,165	692,364	0.12%	16 ms	78 ms
mountain.jpg	1400x1050	1,131,943	1,131,838	0.01%	15 ms	109 ms
mobil.tiff	800x600	655,551	655,379	0.03%	16 ms	62 ms
sawah.tiff	1024x768	1,453,315	1,422,723	2.11%	31 ms	141 ms
monas.tiff	1200x900	1,154,525	1,150,677	0.34%	16 ms	110 ms

Pada pengujian terhadap citra berwarna, secara keseluruhan algoritma huffman dapat mengkompresi setiap format citra yang di uji, walaupun dengan ratio persentase kompresi yang berbeda – beda dan relatif kecil. Seperti yang terlihat pada tabel hasil pengujian diatas. Disini ratio pengujian tertinggi didapat saat menguji objek citra ball.bmp, sebesar 15.36%, hal ini dikarenakan objek citra tersebut memiliki kerapatan intensitas warna yang cukup tinggi pada tiap pikselnya. Ini berarti frekuensi kemunculan tiap – tiap representasi bit yang di miliki citra ini cukup tinggi, hal ini lah yang membuat ratio kompresi yang di dapat cukup besar.

Tabel 2. Citra grayscale setelah di uji Modul Huffman

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
wall.bmp	1600x1200	5,760,054	4,835,770	16.05%	94 ms	437 ms
ubuntu.bmp	800x600	1,440,054	1,372,929	4.67%	16 ms	125 ms
cat.bmp	580x440	765,654	649,602	15.16%	16 ms	63 ms
sky.jpg	1024x768	437,639	438,552	100%	0 ms	47 ms
bambu.jpg	1600x1280	1,247,252	1,246,075	0.1%	31 ms	125 ms
Sungai.jpg	960x750	555,009	554,331	0.13%	0 ms	62 ms
night.tiff	800x600	691,791	692,041	100%	16 ms	63 ms
anime.tiff	1400x1050	1,323,001	1,319,480	0.27%	32 ms	140 ms
jakarta.tiff	600x450	498,081	495,865	0.45%	0 ms	63 ms

Seperti yang terlihat pada tabel hasil pengujian kedua ini, yang dilakukan terhadap citra grayscale. Ratio kompresi yang di hasilkan untuk format *bmp terlihat lebih baik dari pada dengan format yang lainnya. Hal ini sama seperti pada pengujian sebelumnya untuk citra berwarna. Algoritma Huffman cocok dengan format bmp karena merupakan format yang memiliki data mentah dari gambar yang dapat memiliki 24-bit RGB pallate gambar dan skala gambar abu-abu.

Pada tabel pengujian citra grayscale ini, ratio yang di dapat untuk format bmp-nya lebih besar dibandingkan format bmp citra berwarnanya, hal ini dikarenakan, citra grayscale hanya memiliki intensitas derajat keabu - abuan yang tinggi dan tidak mengandung warna lain, oleh karena itu hanya representasi bit dari intensitas derajat ke abu-abuan saja yang di proses dengan jumlah frekuensinya yang tinggi berdasarkan imagenya.

Selain itu, pada tabel pengujian diatas terlihat modul ini tidak dapat mengkompresi untuk citra sky.jpg dan night.tiff. hal ini bisa saja terjadi, karena format jpg itu sebelumnya merupakan format kompresi citra lossy yang intinya memiliki nilai bitrate yang kecil. Sedangkan format tiff hanya menyimpan 16-bit per warna merah, hijau, dan biru untuk total 48-bit atau 8-bit per warna dengan menggunakan nama file. Dimana untuk format tiff, kompresi lossless relatif baik untuk tingkat dua (hitam dan putih, tidak abu-abu). Oleh karena itu pada pengujian grayscale ini, kinerjanya untuk format jpg dan tiff tidak terlalu bagus.

Dan untuk waktu kompresi dan dekompresi, baik itu citra berwarna dan grayscale, secara keseluruhan terlihat bahwa waktu yang dibutuhkan untuk melakukan kompresi lebih cepat di bandingkan dengan waktu yang dibutuhkan untuk melakukan proses dekompresinya. Ini terjadi karena proses penghitungan waktu kompresi didasarkan pada saat pengolahan index dari representasi bit yang mewakili tiap – tiap bit yang dikompresi.

Jadi Huffman menghasilkan jumlah bit yang lebih sedikit untuk penggunaan simbol yang lebih banyak. Sedangkan untuk proses penghitungan waktu dekompresi dilakukan saat melakukan pengecekan kembali tiap- tiap

kode biner bit per bit hasil kompresi untuk di dekompresi kembali. Hal ini lah yang membuat perbedaan waktu dekompresi lebih lama dari pada kompresinya.

Untuk pengujian algoritma LZW, menggunakan objek citra yang sama dengan algoritma Huffman, Dan hasil pengujian terhadap citra berwarna terlihat pada tabel 3 dan tabel 4.

Tabel 3. Citra berwarna setelah di uji modul LZW

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
motor.bmp	1600x1200	5,760,054	3,785,429	34.29%	7500 ms	3813 ms
city.bmp	700x525	1,102,554	1,095,561	0.64%	1891 ms	1078 ms
ball.bmp	950x713	2,033,530	1,274,959	37.1%	2375 ms	1312 ms
heli.jpg	1024x768	287,566	388,903	100%	656 ms	360 ms
bear.jpg	1200x900	693,165	979,939	100%	1625 ms	906 ms
mountain.jpg	1400x1050	1,131,943	1,592,977	100%	2625 ms	1484 ms
mobil.tiff	800x600	655,551	908,451	100%	1500 ms	843 ms
sawah.tiff	1024x768	1,453,315	1,870,303	100%	3094 ms	1781 ms
monas.tiff	1200x900	1,154,525	1,557,473	100%	2547 ms	1468 ms

Pada pengujian terhadap citra berwarna, secara keseluruhan terlihat bahwa algoritma LZW disini lebih baik penggunaannya terhadap citra dengan format bmp dibandingkan dengan format lainnya. Ini karena format bmp merupakan format yang memiliki data mentah dari gambar yang dapat memiliki 24-bit RGB pallate gambar dan skala gambar abu abu. Hal Ini berbeda dengan algoritma Huffman yang cukup baik dalam mengkompresi tiap – tiap format citra yang diuji.

Dalam pengujian kedua ini ternyata algoritma LZW tidak cocok untuk mengkompres format citra jpg dan tiff, karena dalam beberapa ukuran file sampel jenis, menghasilkan ratio kompresi yang bernilai negatif atau dapat ditulis sebagai 100%, Ini berarti tidak terjadi kompresi melainkan pemekaran data karena bernilai negatif. Hal ini dapat terjadi karena format jpg merupakan format citra lossy yang pada intinya memiliki bit rate yang kecil. Sedangkan algoritma LZW merupakan teknik kompresi yang menggunakan prinsip dictionary-based, dimana dictionary based coding tidak efektif digunakan pada input yang datanya pendek karena pada prinsipnya dictionary-based coding merupakan teknik kompresi yang adaptif, sehingga bisa saja hasil kompresi data yang pendek menghasilkan output yang lebih besar. Sedangkan untuk format tiff dapat bersifat lossy atau lossless, jadi ada kemungkinan untuk tidak dapat di kompresi jika itu bersifat lossy.

Untuk performa waktu kompresi dan dekompresi yang di dapat dengan modul LZW ini, ternyata berdasarkan data hasil pengujiannya. Waktu yang dibutuhkan untuk melakukan kompresi lebih besar dibandingkan pada saat dekompresi. Hal ini disebabkan karena pada tahapannya, algoritma LZW berjalan dengan kompleks, dimana dilakukan pengecekkan pada setiap karakter yang muncul kemudian menggabungkan dengan karakter selanjutnya menjadi sebuah string. Dan jika string baru tersebut tidak berada dalam dictionary atau belum diindekskan maka string baru tersebut akan diindekskan ke dalam dictionary dalam bentuk code word. Begitu seterusnya hingga end of file. Hal ini lah yang membuat prosesnya berjalan lama.

Tabel 4. Citra grayscale setelah di uji modul LZW

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
wall.bmp	1600x1200	5,760,054	1,174,945	79.61%	6406 ms	1250 ms
ubuntu.bmp	800x600	1,440,054	322,304	77.62%	1171 ms	422 ms
cat.bmp	580x440	765,654	576,818	24.67%	1031 ms	703 ms
sky.jpg	1024x768	437,639	613,049	100%	1000 ms	562 ms
bambu.jpg	1600x1280	1,247,252	1,788,352	100%	2516 ms	1953 ms
Sungai.jpg	960x750	555,009	788,764	100%	1110 ms	875 ms
night.tiff	800x600	691,791	933,719	100%	1531 ms	875 ms
anime.tiff	1400x1050	1,323,001	1,865,992	100%	2625 ms	2062 ms
jakarta.tiff	600x450	498,081	664,754	100%	937 ms	750 ms

Sesuai dengan hasil pengujian algoritma LZW diatas untuk objek citra grayscale, disini kinerja algoritma LZW sangat bagus pada saat mengkompresi file dengan format bmp, hal ini sama seperti pengujian sebelumnya pada objek citra berwarna. Begitu juga untuk format jpg dan tiff yang pada intinya tidak cocok untuk di terapkan pada modul LZW ini.

Hal ini disebabkan karena format jpg merupakan format citra lossy yang pada intinya memiliki bit rate yang kecil. Sedangkan algoritma LZW merupakan teknik kompresi yang menggunakan prinsip dictionary-based, dimana dictionary-based coding tidak efektif digunakan pada input yang datanya pendek karena pada prinsipnya dictionary-based coding merupakan teknik kompresi yang adaptif, sehingga bisa saja hasil kompresi data yang pendek menghasilkan output yang lebih besar. Sedangkan untuk format tiff dapat bersifat lossy atau lossless, jadi ada kemungkinan untuk tidak dapat di kompresi jika itu bersifat lossy.

Sedangkan untuk performa waktu kompresi dan dekompresi secara keseluruhan untuk ke dua tabel pengujian modul LZW ini, memiliki kesamaan yaitu waktu dekompresi berjalan lebih cepat dari pada saat melakukan kompresi, hal ini karena. Dalam proses dekompresi, algoritma LZW tidak menyimpan string table yang berisi indeks-indeks dari setiap code word yang dihasilkan dalam proses kompresi ke momory akan tetapi menggunakan beberapa informasi yang telah disimpan sebelumnya antara lain 256 karakter ASCII, karakter pertama dari inputan dan code word terakhir.

Pengujian terakhir disini yaitu untuk modul program RLE. Lebih jelasnya dapat dilihat pada tabel 5 untuk objek citra berwarna dan pada tabel 6 untuk hasil pengujian terhadap objek citra grayscale.

Tabel 5. Citra berwarna setelah di uji modul RLE

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
motor.bmp	1600x1200	5,760,054	6,426,169	100%	110 ms	63 ms
city.bmp	700x525	1,102,554	1,217,200	100%	31 ms	15 ms
ball.bmp	950x713	2,033,530	2,221,829	100%	31 ms	15 ms
heli.jpg	1024x768	287,566	322,395	100%	0 ms	0 ms
bear.jpg	1200x900	693,165	777,468	100%	0 ms	16 ms
mountain.jpg	1400x1050	1,131,943	1,269,998	100%	31 ms	15 ms
mobil.tiff	800x600	655,551	908,451	100%	0 ms	0 ms
sawah.tiff	1024x768	1,453,315	1,631,754	100%	32 ms	15 ms
monas.tiff	1200x900	1,154,525	1,295,046	100%	31 ms	15 ms

Pada tabel 5 terlihat bahwa secara keseluruhan algoritma RLE tidak dapat mengkompresi citra berwarna. Bahkan rationya berupa minus dan hasil yang di dapat lebih besar dari sebelumnya, ini berarti terjadi pemekaran data atau dapat dibilang modul RLE ini tidak dapat mengkompresi citra berwarna.

Tabel 6. Citra grayscale setelah di uji modul RLE

File citra	resolusi	memori (bytes)		%	waktu (ms)	
		sebelum	sesudah	Ratio	kompresi	dekompresi
wall.bmp	1600x1200	5,760,054	1,634,671	71.63%	78 ms	109 ms
ubuntu.bmp	800x600	1,440,054	308,792	78.56%	16 ms	16 ms
cat.bmp	580x440	765,654	249,345	67.44%	16 ms	0 ms
sky.jpg	1024x768	437,639	491,059	100%	0 ms	0 ms
bambu.jpg	1600x1280	1,247,252	1,400,041	100%	16 ms	16 ms
Sungai.jpg	960x750	555,009	622,627	100%	16 ms	0 ms
night.tiff	800x600	691,791	776,118	100%	16 ms	0 ms
anime.tiff	1400x1050	1,323,001	1,484,278	100%	32 ms	31 ms
jakarta.tiff	600x450	498,081	558,722	100%	0 ms	0 ms

Kinerja algoritma RLE lebih baik penggunaannya pada format citra bmp yang bersifat grayscale. Hal ini terlihat dari hasil kapasitas kompresi yang dihasilkan pada citra grayscale format bmp yang menjadi lebih kecil serta memiliki ratio yang cukup tinggi yaitu diatas 50%. sedangkan untuk waktu kompresinya memiliki performa yang bagus karena hanya membutuhkan waktu yang sedikit untuk melakukan proses kompresi dan dekompresinya. Hal ini disebabkan karena pada prosesnya algoritma RLE memanfaatkan tingkat korelasi yang tinggi yang terjadi pada bit bit yang berurutan pada perulangan karakter.

4 Kesimpulan

Pada Algoritma Huffman memiliki kompabilitas tinggi untuk dapat mengkompresi setiap format objek citra yang digunakan yaitu bmp, jpg dan tiff. Selain itu algoritma ini juga memiliki kecepatan yang relatif cepat.

Algoritma LZW (Lempel Ziv Welch) menghasilkan performa waktu kompresi dan dekompresi yang kurang baik tetapi algoritma ini memiliki kinerja kompresi yang tinggi pada format citra bmp, tapi tidak untuk format jpg dan tiff.

Algoritma RLE memiliki kompabilitas kemampuan yang tidak lebih baik dari kedua algoritma sebelumnya, Algoritma ini hanya dapat mengkompresi untuk format bmp dengan jenis grayscale saja. Tetapi dalam hal ini performa dan kinerja yang dihasilkan untuk format bmp grayscale cukup tinggi.

References:

[1] Abel, Jurgen, *The Data Compression Resource on the Internet*, http://www.data-compression.info/index.html, diakses tanggal 10 September 2009.

[2] Campos, Arturo San Emeterio, *Run Length Encoding*,http://www.arturocampos.com/ac_rle.html #En coder_implementation, diakses pada tanggal 15 September 2009.

[3] Cary, David, *Beginner Tutorials and Complete Explanation for Data Compression,* http://www.rdrop.com/~cary/html/data_compression .html, diakses pada tanggal 25 Agustus 2009.

[4] Geeks, *C++ Reference, http://www.cppreference.com/wiki/start,* diakses pada tanggal 9 Desember 2009.

[5] Jain, Ramesh, *Machine Vision*, McGraw-Hill, 1995.

[6] Nekrich, Yakov, *Lossless Image Compression*, http://theory.cs.uni-bonn.de/~yasha/limcomplinks.html, diakses pada tanggal 14 September 2009.

[7] Nelson, Mark and Gailly, Jean-Loup, *The Data Compression Book (Second Edition)*, M&T Books, 1996.

[8] Laurens, How *Huffman Compression Works, advantages and Disanvantages,* http://www.prepressure.com/library/compression_algorithms/huffman, diakses pada tanggal 10 September 2009.

[9] Pitas, Ioannis, *Digital Image Processing Algorithms,* Prentice – Hall International, 1993.

[10] Renaldi Munir, *Pengolahan Citra Dengan Pendekatan Algoritmatik,* INFORMATIKA, Bandung, 2004.

[11] Salomon, David, *Data Compression; The Complete Reference (Second Edition)*, Springer Verlag, 2000.

[12] Schildt, Herbert, *C++ : The Complete Reference*, McGraw-Hill, 1998.