Perbandingan Algoritma Kompresi Terhadap Objek Citra Menggunakan JAVA

Maria Roslin Apriani Neta
Program Studi Magister Teknik Informatika, Universitas Atma Jaya Yogyakarta
Jl. Babarsari no 43 55281 Yogyakarta Telp (0274)-487711
e-mail : aprianyneta@yahoo.co.id[1)],

ABSTRAK

Dalam ilmu komputer, pemampatan data atau kompresi data adalah sebuah cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpanan lebih kecil sehingga lebih efisien dalam menyimpannya atau mempersingkat waktu pertukaran data tersebut. Kebalikan dari proses kompresi adalah dekompresi, yang merupakan proses pengembalian data yang telah dihasilkan oleh proses kompresi. Apabila hasil dekompresi menghasilkan data yang sama persis dengan data aslinya yang dikompresi, maka kompresi tersebut disebut lossless compression. Sebaliknya, jika hasil dekompresi meghasilkan data yang tidak sama dengan data aslinya yang dikompresi, karena ada data yang hilang atau dianggap tidak terlalu penting mengubah informasi, maka disebut lossy compression. Dalam paper ini akan dibahas mengenai perbandingan empat algoritma proses kompresi yaitu algoritma Huffman, LZW, RLE dan Shannon-Fano. Pada dasarnya keempat algortima ini sama membentuk suatu kode yang memiliki redundansi minimum sehingaa manajemen memori dapat dilakukan dengan baik. Namun setiap algoritma memilki kompleksitas yang berbeda-beda dalam hal kecepatan dan kapasitas dalam pengkompresi dan dekompresi data. Untuk memperoleh hasil uji sebagai perbandingan kompresi objek digital, maka modul program dibuat menggunakan bahasa JAVA dimana hasilnya akan menentukan mana algoritma yang terbaik dengan menghasilkan ukuran kompresi yang lebih kecil dari keempat algoritma yang di ujikan.

Kata kunci : *Algoritma, Kompresi, Dekompresi, Java, Objek Digital*

1. PENDAHULUAN

Perkembangan teknologi informasi yang pesat sangat memperngaruhi dalam penyajian sebuah data atau informasi. Yang mana tidak hanya dalam bentuk teks, tetapi sekarang sebuah data atau informasi dapat berupa *image, audio* dan video. Keempat elemen data tersebut merupakan bagian dari objek digital. Pada umumnya representasi objek dalam bentuk digital membutuhkan memori. Semakin besar ukuran objek, maka semakin besar pula memori yang dibutuhkan. Pada saat ini, kebanyakan aplikasi menginginkan representasi data dengan kebutuhan memori yang kecil. Kompresi merupakan proses untuk 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 menjadi pertimbangan dalam memilih suatu metode kompresi, yaitu kecepatan kompresi, sumber daya yang dibutuhkan, ukuran file hasil kompresi serta kompleksitas algortima. Prinsip umum yang digunakan dalam kompresi citra adalah mengurangi redudansi dari data atau informasi yang terdapat dalam sebuah objek digital sehingga dapat disimpan dan ditransmisi dengan efisien.

Kebalikan dari proses kompresi data yaitu proses dekompresi. Dekompresi adalah sebuah proses untuk mengembalikan data baru yang telah dihasilkan oleh proses kompresi menjadi data awal. Dekompresi yang menghasilkan data sama persis dengan data aslinya sebelum kompresi, maka data tersebut disebut *lossless compression.* Sebaliknya, jika hasil dekompresi menghasilkan data tidak sama persis dengan data aslinya sebelum dekompresi, karena ada data yang dihilangkan karena dirasa tidak terlalu penting tetapi tidak mengubah informasi yang dikandungnya, disebut *lossy compression.*

Paper ini akan mencoba untuk menganalisa perbandingan antara empat algoritma kompresi yaitu huffman, Lampe-Ziv-Welch (LZW), Run Length Encoding (RLE) dan Shannon-Fano untuk mengetahui ratio kompresi perbandingan memori sebelum dan sesudah dilakukan kompresi.

2. TINJAUAN PUSTAKA

2.1. Pengertian Kompresi

Kompresi berarti memampatkan atau mengecilkan ukuran, kompresi data adalah suatu proses mengkodekan informasi menggunakan bit atau *information-bearing* unit yang lain yang lebih rendah dari pada representasi data yang tidak terkodekan dengan suatu sistem encoding tertentu dan kompresi data adalah suatu cara untuk memadatkan data sehingga hanya memerlukan ruangan penyimpanan lebih kecil sehingga lebi efisien dalam penyimpanannya dan mempersingkat waktu pertukaran data tersebut. Keuntungan kompresi data adalah penghematan tempat pada media penyimpanan dan penghematan *bandwitch* pada pengiriman data. Namun kompresi memiliki sisi negatife, bila data yang dikompresi akan dibaca maka harus dilakukan proses dekompresi terlebih dahulu.[2]

2.2. Pengertian Algoritma Huffman

Algoritma Huffman adalah salah satu algoritma kompresi tertua yang disusun oleh David Huffman pada tahun 1952. Algoritma tersebut digunakan untuk membuat kompresi jenis lossy compression, yaitu pemampatan data dimana tidak satu byte pun hilang sehingga data tersebut utuh dan disimpan sesuai dengan aslinya. Pada sejarahnya, Huffman sudah tidak dapat membuktikan apapun tentang kode apappun yang efisien, tapi ketika tugasnya hamper final, ia mendapatkan ide untuk menggunakan pohon binary untuk menyelesaikan masalahnya mencari kode yang efisien. Pada dasarnya, algoritma Huffman ini bekerja seperti mesin sandi morse, dia membentuk suatu kode dari suatu karakter. Sehingga karakter tersebut memiliki rangkaian bit yang lebih pendek dibandingkan sebelumnya.[7]

2.3. Pengertian Algortima Lempel-Ziv-Welch (LZW)

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.[7]

2.4. Pengertian Algoritma Run-Length (RLE)

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 Runlength paling efektif pada file – file grafis, dimana biasanya berisi deretan karakter yang sama. (Wijaya and Widodo 2010)

2.5. Pengertian Algortima Shannon-Fano

Algoritma Shannon-Fano coding ditemukan oleh Claude Shannon (bapak teori informasi) dan Robert Fano pada tahun 1949. Pada saat itu metode ini merupakan metode yang paling baik tetapi hamper tidak pernah digunakan dan dikembangkan lagi setelah kemunculan algoritma Hufman. Pada dasarnya metode ini menggantikan setiap simbol dengan sebuah alternatif kode biner yang panjangnya ditentukan berdasarkan probabilitas dari simbol tersebut Di bidang kompresi data, Shannon-Fano coding adalah teknik untuk membangun sebuah kode awalan didasarkan pada seperangkat simbol dan probabilitas (diperkirakan atau diukur). Namun, algoritmya ini dirasa kurang optimal dalam arti bahwa ia tidak mampu mencapai kode seefisien mungkin seperti kode diharapkan panjang seperti algoritma Huffman. [1]

3. PERANCANGAN SISTEM

3.1. Flowchart Algoritma Huffman

Tahapan proses kompresi algoritma Huffman :

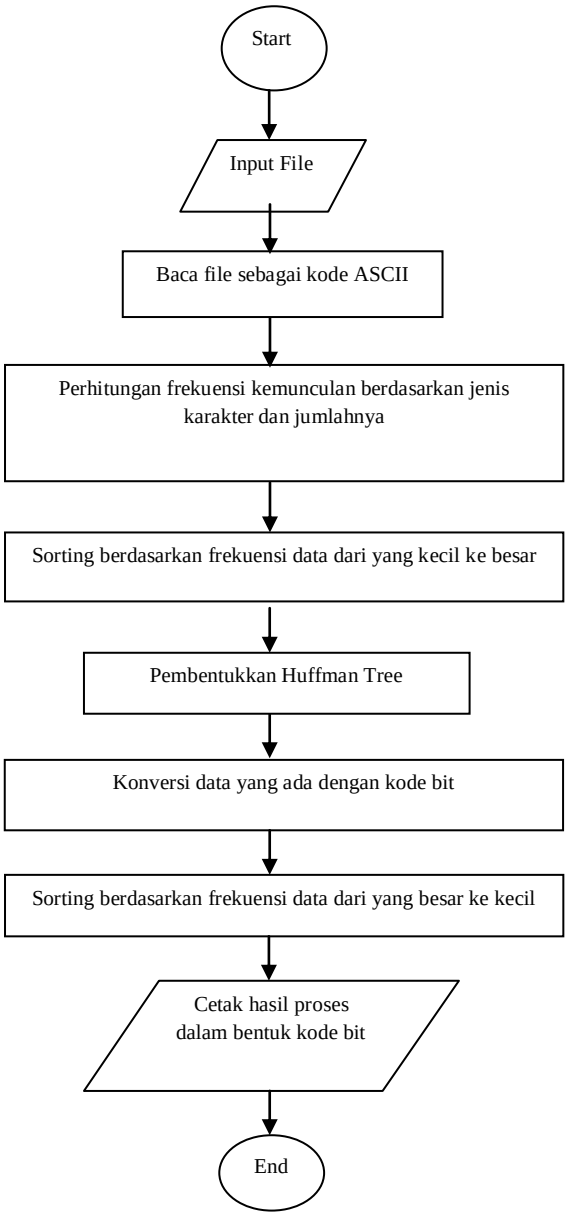
a. Hitung banyaknya jenis karakter dan jumlah dari masing-masing karakter yang terdapat dalam sebuah file.
b. Susun setiap jenis karakter dengan urutan jenis karakter yang jumlahnya paling sedikit ke yang jumlahnya paling banyak.
c. Buat pohon biner berdasarkan urutan karakter dari yang jumlahnya terkecil 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.

3.2. Flowchart Algortima Lempel-Ziv-Welch (LZW)

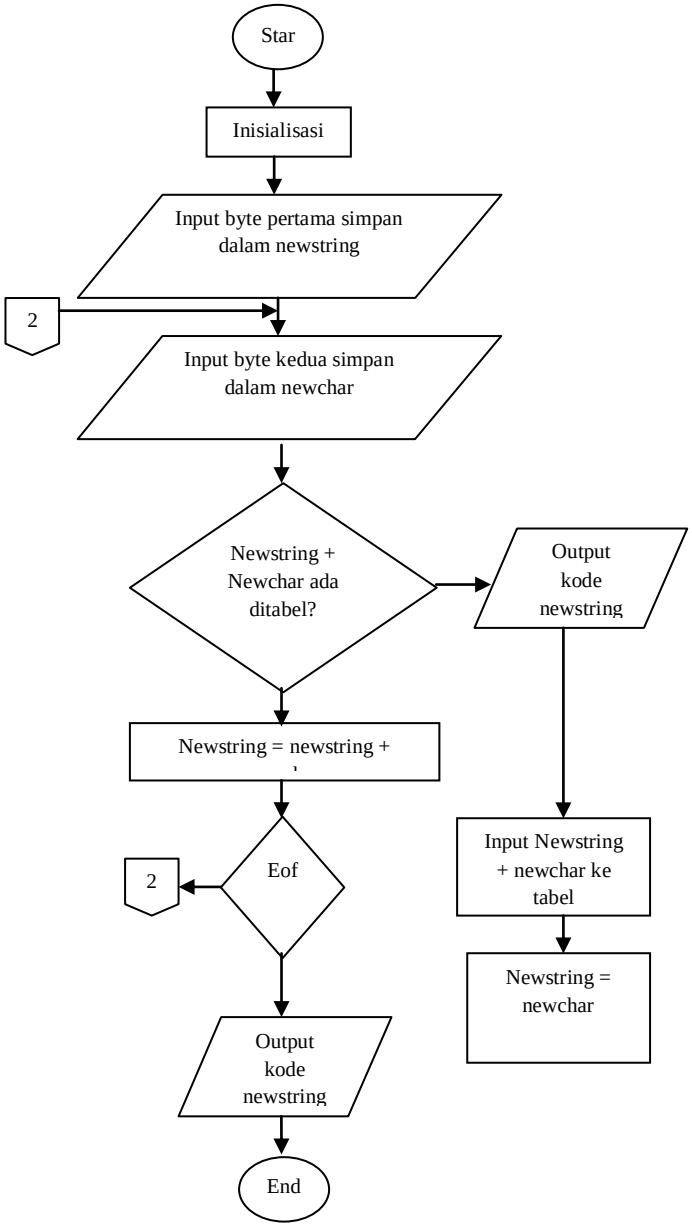
Tahapan proses dalam algoritma LZW :

a. Dictionary diinisialisasi dengan semua karakter dasar yang ada {'A'..'Z','a'..'z','0'..'9'}.
b. P <- input pertama dalam *stream* newstring.
c. Q = input berikutnya dalam *stream* newchar.
d. Apakah *string* (*P* + *Q*) terdapat dalam *dictionary* ?
   i. Jika ya, maka *P* = *P* + *Q* (gabungkan *P* dan *Q* menjadi *string* baru).
   ii. Jika tidak, maka ;
      - *Output* sebuah kode untuk menggantikan *string P*.
      - Tambahkan *string* (*P* + *Q*) ke dalam *dictionary* dan berikan nomor/kode berikutnya yang belum digunakan dalam *dictionary* untuk *string* tersebut. *P* = *Q*
e. Apakah masih ada karakter berikutnya dalam *stream* karakter ?

i. Jika ya, maka kembali ke langkah 2.
ii. Jika tidak, maka *output* kode yang menggantikan *string a*, lalu terminasi proses (*stop*).



Gambar 3.1 *Flowchart* Algoritma Huffman

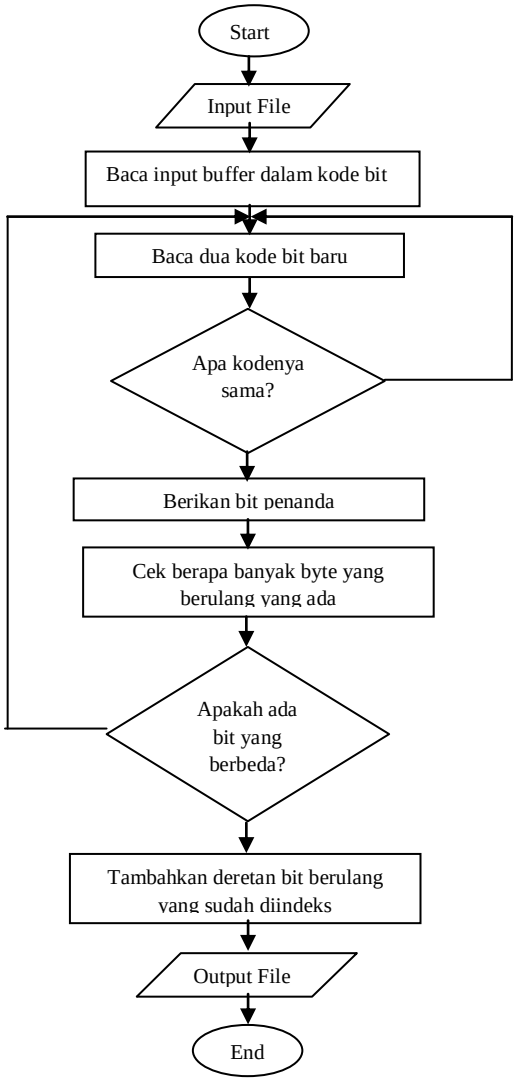


Gambar 3.2 *Flowchart* Algoritma LZW

3.3. Flowchart Algortima Run-Length (RLE)

Tahapan dalam algoritma RLE :

a. 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.
b. 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.
c. Tambahkan deretan bit untuk menyatakan jumlah karakter yang sama berurutan, pada contoh di atas karakter yang sama berturutan sebanyak delapan kali, jadi diberikan deretan bit 00001000 (8 desimal).
d. Tambahkan deretan bit yang menyatakan karakter yang berulang, misal pada contoh di atas karakter yang berulang adalah 01000001 atau karakter A pada karakter ASCII.



Gambar 3.3 *Flowchart* Algoritma RLE

3.4. Flowchart Algortima Shannon –Fano

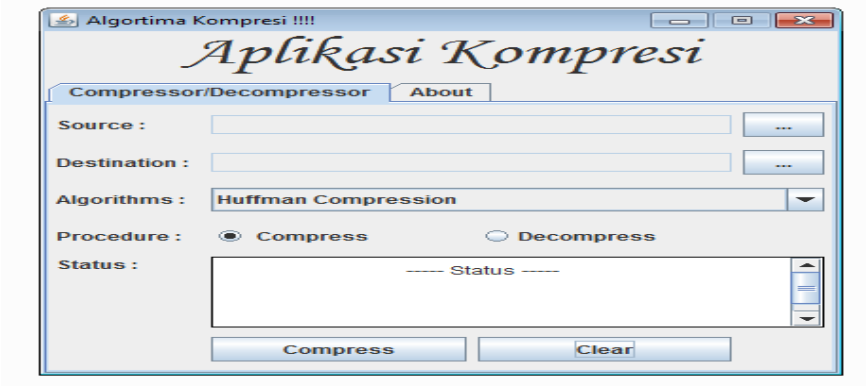
Tahapan dalam algoritma Shannon-Fano :

a. Untuk daftar simbol-simbol tertentu, mengembangkan sebuah daftar yang sesuai probabilitas atau frekuensi relative kejadian diketahui.
b. Menyortir daftar simbol-simbol sesuai dengan frekuensi, dengan simbol-simbol yang paling sering terjadi disebelah kiri dan yang paling umum disebelah kanan.
c. Membagi daftar menjadi dua bagian, dengan total frekuensi dihitung dari kiri setengah menjadi seperti dekat dengan jumlah yang tepat mungkin.
d. Kiri setengah dari daftar diterapkan angka biner 0, dan hak diberikan setengah digit 1. Ini berarti bahwa kode untuk simbol-simbol pada semester pertama semua akan dimulai dengan 0, dan aturan-aturan diparuh kedua akan semua mulai dengan 1.
e. Rekursif menerapkan langkah 3 dan 4 untuk masing-masing dari dua bagian, membagi keompok dan menambahkan kode bit sampai setiap simbol telah menjadi kode yang sesuai daun dipohon.

4. IMPLEMENTASI DAN PENGUJIAN SISTEM

4.1. Implementasi Sistem

Implementasi dari paper ini dibuat berdasarkan hasil analisa dalam perancangan sistem yang dikembangkan menggunakan bahasa pemrograman JAVA. Aplikasi kompresi yang membandingkan beberapa metode yang akan digunakan, dapat dilihat pada gambar 4.1



Gambar 4.1 Aplikasi Kompresi

Untuk mengetahui tingkat perbandingan dari metode tersebut, maka dilakukan pengujian sebanyak tiga model. Pertama, menggunakan beberapa citra dengan warna yang berbeda dengan resolusi yang sama. Kedua, menggunakan beberapa citra dengan warna yang sama dan resolusi yang berbeda. Ketiga, menggunakan beberapa citra dengan *full colors.*

4.2. Pengujian Sistem

a. Pengujian kompresi Citra dengan Jumlah warna yang berbeda dan resolusi yang sama

Warna yang terlihat pada citra *bitmap* merupakan perpaduan dari tiga warna dasar, yakni merah (R), Hijau (G), Biru (B). resolusi citra yang dipakai 64 x 64 *Pixel* dengan warna yang berbeda-beda.

Tabel 4.1 Algo. Huffman uji 1

Jumlah Warna	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
1	12342	2102	5.871	82.96%
2	12342	2651	4.65	78.52%
3	12344	3064	4.028	75.17%
4	12344	3323	3.71	73.08%

Tabel 4.2 Algo LZW uji 1

Jumlah Warna	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
1	12342	342	36.08	97.22%
2	12342	555	22.23	95.50%
3	12344	656	18.81	94.68%
4	12344	756	16.37	93.89%

Tabel 4.3 Algo RLE uji 1

Jumlah Warna	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
1	12342	16427	0.751	-33.09%
2	12342	12331	1.000	0.089%
3	12344	12340	1.000	0.032%
4	12344	12340	1.000	0.032%

Tabel 4.4 Algo Shannon – Fano uji 1

Jumlah Warna	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
1	12342	3644	3.38	70.47%
2	12342	4192	2.94	66.034%
3	12344	4606	2.67	62.68%
4	12344	4864	2.53	60.59%

Di atas ini merupakan hasil uji perbandingan algoritma untuk masing-masing citra dengan warna yang berbeda tapi memiliki resolusi yang sama. Untuk memperoleh hasil dari pengujian, penulis menggunakan beberapa citra dengan jumlah warna sebanyak 4 resolusi sebesar 64 x 64. Dari hasil tersebut dapat disimpulkan bahwa algoritma LZW (*lemple –Ziv-Welch)* lebih baik dari pada ketiga metode lainnya. Hasil kompresi lebih kecil sehingga memori yang dipakai tidak terlalu banyak. Dan algortima RLE (*Run-Length*) dianggap kurang baik karena hasil kompresinya lebih besar sehingga memakan memori.

b. Pengujian kompresi citra dengan resolusi yang berbeda

Resolusi dari suatu citra sangat mempengaruhi ukuran dari citra tersebut. Semakin besar resolusi maka ukuran citra akan semakin besar dan memori yang dipakai akan semakin besar. Dalam pengujian ini warna yang dipakai sama yaitu empat warna, tetapi resolusi akan berbeda. Hasilnya dapat dilihat pada table di bawah ini.

Tabel 4.5 Algo. Huffman uji 2

Resolusi Citra	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
64 x 64	12344	3323	3.71	73.08%
128 x 128	49208	14982	3.284	69.55%
256 x 256	196664	61202	3.21	68.87%
512 x 512	786488	247675	3.17	68.50%
1024 x 1024	3145784	985483	3.19	68.67%

Tabel 4.6 Algo LZW uji 2

Resolusi Citra	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
64 x 64	12344	754	16.371	93.89%
128 x 128	49208	2463	19.97	94.99%
256 x 256	196664	8348	23.55	95.75%
512 x 512	786488	289478	2.71	63.19%
1024 x 1024	3145784	1151777	2.73	63.38%

Tabel 4.7 Algo RLE uji 2

Resolusi Citra	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
64 x 64	12344	12340	1.000	0.032%
128 x 128	49208	50606	0.97	-2.84%
256 x 256	196664	200978	0.978	-2.19%
512 x 512	786488	800265	0.982	-1.751%
1024 x 1024	3145784	3206967	0.980	-1.944%

Tabel 4.8 Algo Shannon – Fano uji 2

Resolusi Citra	Citra Asli	Hasil Kompresi	Rasio Kompresi	Persentase Kompresi
64 x 64	12344	4864	2.53	60.59%
128 x 128	49208	19807	2.48	59.74%
256 x 256	196664	80390	2.44	59.123%
512 x 512	786488	324362	2.424	58.75%
1024 x 1024	3145784	1295342	2.42	58.82%

Di atas merupakan hasil perbandingan dari beberapa algoritma dengan pengujian berbeda resolusi menggunakan citra yang sama. Dari hasil table perbandingan di atas diketahui algoritma LZW lebih baik untuk model pengujian kompresi beda resolusi namun memiliki citra yang sama. Dengan hasil algoritma LZW ini memori yang akan terpakai hanya sedikit jika dibandingkan dengan algoritma RLE yang memakan memori cukup besar. Dari hasil persentasi kompresi RLE terdapat persentasi negatif yang artinya hasil kompresi lebih besar dari ukuran citra asli. Algoritma LZW lebaih baik digunakan karena tidak memakan memori yang banyak.

c. Pengujian Kompresi citra dengan *full colors*

Dalam pengujian ini digunakan beberapa gambar yang biasa digunakan untuk uji coba pengolahan citra. Dalam pengujian ini terdapat tiga citra seperti pada table di bawah ini.

Tabel 4.9 Citra Uji

Nama File	Citra BMP
Lena.bmp	
Baboon.bmp	
Papper.bmp	

Hasil keseluruhan dari pengujian dari ke empat algoritma tersebut dapat dilihat pada table di bawah ini.

Tabel 4.10 Hasil Pengujian Citra

Citra Uji	Ukuran Citra	Algoritma Huffman			Algoritma LZW			Algoritma RLE			Algoritma Shannon -Fano		
		Hasil	Ratio	Persen	Hasil	Ratio	Persen	Hasil	Ratio	Persen	Hasil	Ratio	Persen
Lena.bmp	786486	765187	1.02	2.7%	729326	1.07	7.26%	787245	0.99	-0.09%	783118	1.004	0.428%
Baboon.bmp	786488	767626	1.02	2.3%	745551	1.05	5.20%	786482	1.00	0.0007%	790223	0.99	-0.4%
Papper.bmp	786486	757065	1.03	3.74%	725496	1.08	7.75%	786472	1.00	0.007%	777754	1.01	1.11%

Dari pengujian model ini dapat ditarik kesimpulan bahwa kualitas warna dari suatu citra sangat mempengaruhi ukuran citra yang dikompresi. Semakin banyak perpaduan warna maka akan semakin berbeda hasil kompresinya walaupun ukuran dari citra asli sama. Dengan begitu tetap diketahui bahwa algoritma LZW tetap yang terbaik dengan memberikan nilai hasil kompresi yang lebih kecil sehingga tidak terlalu banyak memori yang dibutuhkan untuk penyimpanannya.

5. PENUTUP

Dari hasil analisa uji coba citra di atas dapat ditarik kesimpulan bahwa pengujian kompresi citra dengan beberapa model uji menghasilkan nilai kompresi menggunakan algoritma LZW lebih kecil dibanding ketiga algoritma yang dipakai. Dengan semakin kecilnya ukuran citra hasil kompresi maka akan semakin kecil kapasitas memori yang digunakan. Salain itu perpaduan warna dalam sebuah citra dapat mempengaruhi ukuran kompresi dari suatu citra.

DAFTAR PUSTAKA

[1] Adhitama, Gagarin. "Perbandingan Algoritma Huffman dengan Algoritma Shannon-Fano." 2010.
[2] Faradisa, Irmalia Suryani, and Bara Firmana Budiono. "Implementasi Metode HUFFMAN sebagai Teknik Kompresi CItra." *Elektro Eltek*, 2011: 176-182.
[3] Linawati, and Panggabean. "Perbandingan Kinerja Algoritma Kompresi Huffman, LZW dan DMC pada Berbagai Tipe File." *Integral*, 2004: 7-16.
[4] Pitas, Ioannis, *Digital Image ProcessingAlgorithms* Prentice - Hall International, 1993.
[5] Putra, Darma. *Pengolahan CItra Digital.* Yogyakarta: Andi Publisher, 2009.
[6] Renaldi, Munir. *Pengolahan Citra Dengan Pendekatan Algoritmatik.* Bandung: INFORMATIKA, 2004.
[7] Wijaya, Aditya, and Suryarini Widodo. "Kinerja dan Performa Algoritma Kompressi Lossless terhadap Objek Citra Digital." 2010.