PEMAMPATAN DATA *LOSSLESS* DENGAN METODE *STATIC-ADAPTIVE ARITHMETIC CODING*

Hernawan Sulistyanto

Teknik Informatika, Fakultas Komunikasi dan Informatika

Universitas Muhammadiyah Surakarta

Email : hnstyanto@yahoo.com

Abstract :

This research explains about some popular algorithms of lossless compression that more people have implemented it in their programs, and how to join them to become a library file which use one of the features of object oriented programming. Implementation of Arithmetic Coding algorithm uses array of both low and high range. Low range array saves the lowest limit for interval of alphabet whereas high range saves limit highest interval. To test the library (the proof of concept), some application programs that use (call) the routines of lossless compression in the library are created to compress-decompress any file types. So the goal of this research is acquire a library file that contain all of the lossless compression algorithms, especially an application program is like toolbox of lossless compression, which is used to test the performance of the library or to support the advanced research (such as cascaded compression, etc.).The results of this research are a library file (static and dynamic) that contain some routines of lossless compression algorithms, namely Static Huffman order-0, Adaptive Huffman order-0,1, Run-Length Encoding order 0,1,2, and 3, LZW, Static Adaptive Aithmetic Coding. Totally, there are four kinds of lossless compression algorithm. In conclusion, the Arithmetic Coding algorithm is the best method based on the performance of compression ratio at 73% , then LZW algorithm, Huffman algorithm and RLE algorithm gain compression ratio 69%, 49% and 19%., respectively

Keywords : data compression, lossless, static library, dynamic link library.

PENDAHULUAN

Perkembangan teknologi informasi berjalan seiring dengan perkembangan kapasistas dan laju data yang ditanganinya. Permasalahan utama penanganan data pada teknologi informasi saat ini adalah bagaimana data informasi yang semakin hari semakin bertambah besar dan kompleks tersebut dapat ditransfer dalam suatu jaringan komputer secara cepat, efisien dan akurat. Perangkat lunak-perangkat lunak aplikasi yang memakai dan mengolah data besar seperti misalnya perangkat lunak untuk manajemen basis data dan *games*, semakin hari semakin besar, komplek dan canggih. Akibatnya kebutuhan akan tempat dalam media penyimpan untuk menyimpan data dari program-program aplikasi tersebut akan terus

bertambah besar. Salah satu solusinya adalah dengan cara memampatkan (*compressing*) data tersebut menjadi lebih kecil dibanding ukuran semula atau menjadi data yang termampatkan. Selanjutnya data yang sudah termampatkan itu baru disimpan dalam media penyimpanan data atau dikirimkan ke lain tempat dalam suatu jaringan komputer. Apabila di lain waktu data tersebut akan digunakan atau diproses untuk keperluan tertentu, data yang termampatkan tadi dimekarkan (*decompresing*) kembali menjadi data aslinya (seperti sebelum dimampatkan) dengan tanpa mengurangi atau menghilangkan sedikitpun isi dari informasi tersebut.

Berdasar latar belakang tersebut maka dirumuskan permasalahan yaitu bagaimana mengimplementasikan algoritma *static-adaptive Arithmetic Coding* untuk pemampatan data, bagaimana mengimplementasikan *Arithmetic Coding* dalam suatu *toolbox* pemrograman untuk mendapatkan *file* terkompresi dan dekompresi, seberapa besarkah prosentase rasio kompresi yang diperoleh pada implementasi konstruksi *toolbox* kompresi.

Maksud dan tujuan dari penelitian ini adalah merancang algoritma *static-adaptive* Arithmatic Coding; membuat konstruksi *toolbox* kompresi yang mengimplementasikan *static-adaptive Arithmaetic Coding*, mencari rasio kompresi data tak hilang dengan *toolbox* tersebut guna dibandingkan dengan algoritma-algoritma yang telah ada.

METODE PENELITIAN

Design program Algoritma *Static Arithmetic Coding*

Algoritma *Arithmetic Coding* tak-adaptif tidak terlalu berbeda dengan algoritma tak-adaptif yang lain. Algoritma ini dimulai dengan menghitung frekuensi kemunculan semua simbol yang ada dalam aliran data masukan (*file* sumber). Implementasi kompresi dengan metode *Arithmetic Coding* memakai dua larik bantuan, yaitu *low_range* dan *high_range*. Larik *low_range* menyimpan batas bawah untuk interval bagian huruf dalam interval 0,0 – 1,0, sedangkan larik *high_range* menyimpan nilai batas maksimumnya.

Algoritma dekompresi *Arithmetic Coding* akan membalik perhitungan algoritma kompresi di atas. Ide dekompresi adalah perhitungan interval antara *LOW* dan *HIGH* seperti dalam proses kompresi, kemudian menghitung interval bagian masing-masing untuk *LOW* dan *HIGH* tersebut. Apabila nilai bilangan masukan jatuh dalam interval bagian untuk huruf ‘c’, maka huruf ‘c’ ini akan dituliskan ke aliran keluaran (*output file*). Selanjutnya *LOW* dan *HIGH* akan disesuaikan seperti halnya dalam proses kompresi. Algoritma dekompresi dapat mengetahui tabel informasi statistik berdasarkan hasil pembacaan *header file* dari *file* masukan (terkompres). Untuk aliran masukan yang besar, presisi yang diperlukan untuk variabel *LOW* dan *HIGH* akan menjadi semakin besar. Akibatnya presisi yang disediakan dalam prosesor komputer nyata tidak mencukupi.

Algoritma *Arithmetic Adaptive Coding*

Pada prinsipnya algoritma *Arithmetic Coding* adaptif sama dengan semua algoritma kompresi adaptif. Sesudah tahap pengkode-an, statistik dan kode harus dihitung kembali. Algoritma *Arithmetic Coding* merubah kode sama dengan perubahan probabilitas untuk semua huruf. Akibatnya juga terjadi perubahan pemecahan interval 0.0 - 1.0 menjadi bagian interval yang lebih kecil.

Urut-urutan algoritma kompresi *Arithmetic Coding* yang adaptif pada dasarnya dapat dijelaskan sebagai berikut.

1. Inisialisasi model.
2. Baca sebuah simbol dari *file* masukan (*file* yang akan dimampatkan). Jika simbol terbaca adalah karakter *End-Of-File* maka program selesai.
3. Enkode simbol yang terbaca tersebut, dengan ‘model saat itu’. Hasilnya yang berupa ‘kode saat itu’ dari simbol tersebut, disimpan langsung ke *file* keluaran.
4. Perbarui (*update*) model.
5. Kembali ke langkah 2.

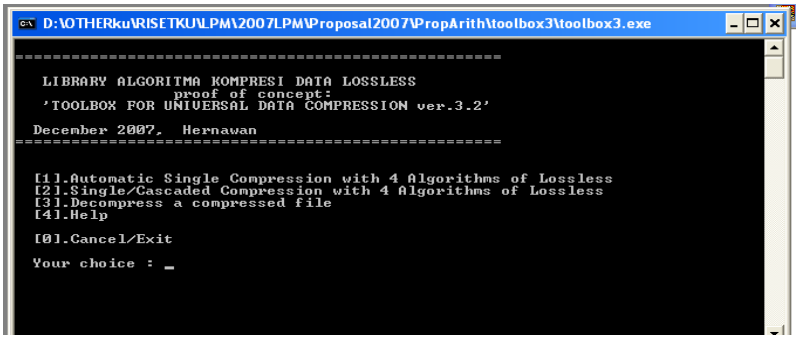
Perlu digarisbawahi di sini adalah pada langkah *update* model karena sesungguhnya proses yang terjadi dalam langkah tersebut amat komplek. Pertama,

algoritma harus menghitung ulang statistik simbol yang muncul, kemudian menghitung probabilitas dan prosentase seluruh simbol yang sudah muncul saat itu. Selanjutnya memperbaharui pembagian interval : 0.0 – 1.0.

HASIL PENELITIAN DAN PEMBAHASAN

Tahapan terakhir dalam penelitian ini adalah program aplikasi *toolbox* kompresi *lossless* tersebut yang dipakai untuk meneliti unjuk kerja setiap algoritma kompresi *lossless* yang tercakup dalam pustaka. Mekanismenya adalah dengan melakukan proses kompresi dan dekompresi terhadap berbagai macam jenis *file* yang umum dipergunakan, misalnya doc, exe, bmp, html, wav, dan *file-file* grafik. Lalu diamati dan dicatat besarnya rasio kompresi dan waktu kompresi (dekompresi) yang diperlukan oleh masing-masing algoritma kompresi.

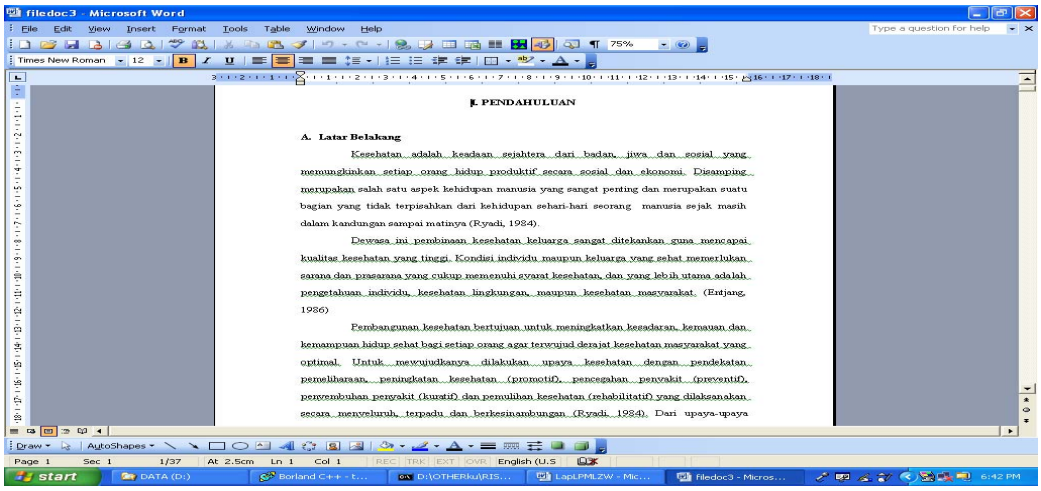
Saat toolbox35.exe dieksekusi maka pertama kali akan tertampil menu utama seperti dalam Gambar 1. Pengguna dapat memilih 5 macam menu utama yang berbeda dengan cara mengetikkan nomor dalam tanda kurung kurawal di depan menu pilihan.



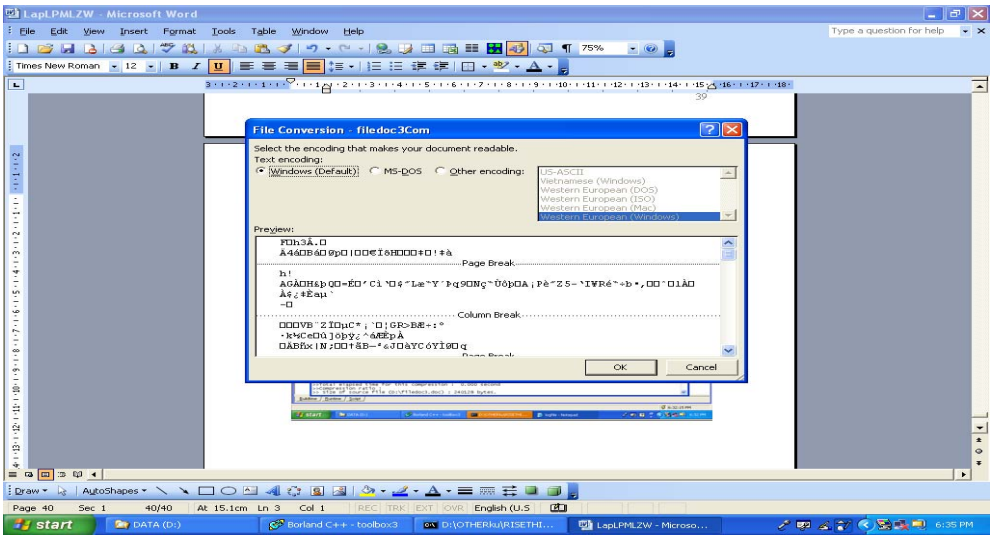
Gambar 1 Menu utama program *toolbox* algoritma kompresi *lossless* versi 3.5

Pengujian program dan pustaka dilakukan dengan cara langsung menggunakan program aplikasi *toolbox35.exe* tersebut untuk mengkompres berbagai tipe dan ukuran *file* sampel. Beberapa buah *file* dengan tipe yang berbeda (yang umum digunakan) dipakai untuk sampel dalam penelitian ini adalah sebagai berikut. Jumlah *file* sampel untuk masing-masing tipe *file* minimal tiga buah dengan ukuran *file* yang bervariasi. Berikut daftar nama (dan tipe *file*) beserta

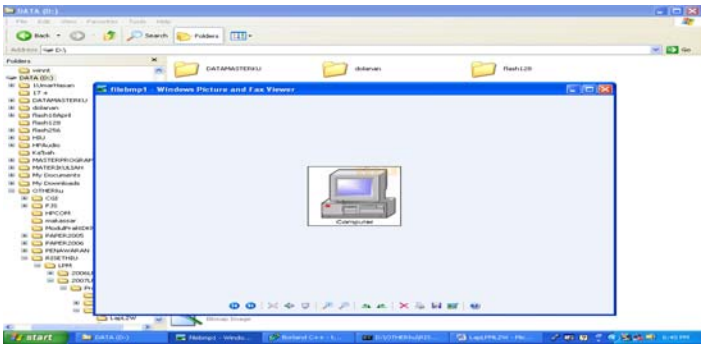
ukurannya. Beberapa perbandingan antara *file* yang belum terkompresi dengan *file* terkompresi mengunakan static-adaptive Arithmetic Coding ditampilkan pada gambar di bawah ini.



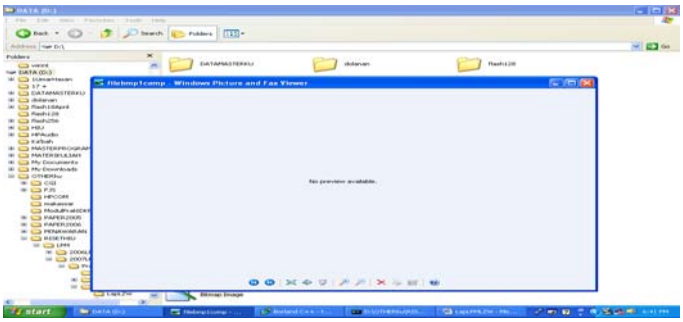
Gambar 2 *File* *.doc sebelum kompresi



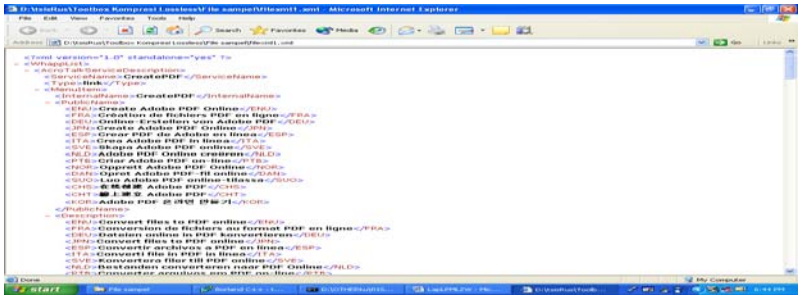
Gambar 3 *File* *.doc hasil kompresi



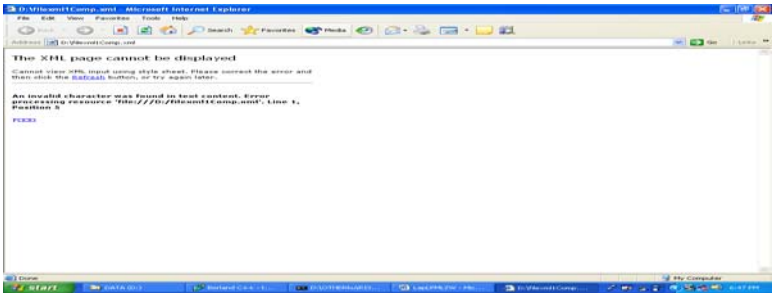
Gambar 4 Isi *file* *.bmp sebelum terkompresi



Gambar 5 Isi *file* *.bmp setelah proses kompresi



Gambar 6 Isi *file* *.xml sebelum terkompresi



Gambar 7 Isi *file* *.xml hasil kompresi

Untuk membantu menganalisa unjuk kerja semua algoritma kompresi *lossless* yang ada dalam pustaka tersebut, maka data hasil penelitian unjuk kerja tiap-tiap algoritma yang diperoleh disajikan dalam bentuk grafik-grafik sebagai berikut.

Gambar 8 Grafik rerata unjuk kerja (rasio kompresi) dari semua algoritma kompresi *lossless* yang ada dalam pustaka ketika dipakai untuk mengkompres *file-file* sampel.

Berdasarkan grafik rerata rasio kompresi yang dihasilkan dapat disimpulkan bahwa algoritma Adaptive Arithmethic Coding memiliki rasio kompresi tertinggi diatas 81% untuk *file-file* bmp, cpp, txt, dan xml. Rerata rasio kompresinya pada keseluruhan *file* sampel adalah 73%. Sedangkan untuk algoritma lain yaitu RLE rerata rasio kompresinya 19% dengan rasio kompresi tertinggi sebesar 56% pada kompresi *file* *.bmp. Huffman memberikan rerata rasio kompresi 49% dengan rasio tertinggi 70% pada kompresi *file* *.bmp. LZW menghasilkan rerata rasio kompresi 69% dengan rasio kompresi tertinggi pada kompresi *file* *.bmp sebesar 84%.

Selanjutnya parameter kedua yang digunakan untuk meneliti unjuk kerja algoritma-algoritma kompresi *lossless* yang ada dalam pustaka tersebut adalah waktu kompresi atau kecepatan kompresi. Data yang diperoleh dari hasil eksekusi program *toolbox35.exe* adalah data-data numerik *(integer)* dalam satuan detik, sehingga untuk mendapatkan kecepatan kompresi harus dilakukan perhitungan lagi dengan membagi ukuran *file* sumber (dalam satuan *byte*) dengan waktu

kompresi yang diperoleh tersebut, sehingga dihasilkan nilai dari kecepatan kompresi dalam satuan *byte* (atau kilo *byte*) per detik (*bps/kbps)*. Beberapa algoritma seperti *RLE* dan *LZW* tidak diperoleh data yang lengkap untuk seluruh *file* sampel dalam satu tipe (karena kecepatan kompresi yang tinggi), sehingga perhitungan kecepatan kompresi dilakukan berdasarkan waktu kompresi rata-rata untuk satu jenis *file* sampel.

Gambar 9 Kecepatan kompresi untuk masing-masing algoritma dalam *Kbps*

Berdasar grafik hasil penelitian tersebut di atas, dapat dianalisa bahwa secara umum algoritma yang berunjuk kerja rasio kompresi lebih tinggi, akan membawa konsekuensi waktu kompresi yang relatif lebih lama (kecepatan kompresi rendah), atau dengan kata lain, rasio kompresi dan kecepatan kompresi saling berbanding terbalik.

Gambar 10 Grafik kecepatan rata-rata dekompresi untuk masing-masing algoritma

Berdasarkan grafik kecepatan kompresi dan dekompresi, algoritma yang paling cepat waktu kompresi dan dekompresi adalah *RLE* orde-2 dengan rate 984164 kbyte/s pada kompresi *file* *.txt, lalu *RLE* orde-3, dan *RLE* orde-0. Rerata kecepatan kompresi RLE tertinggi adalah 622687 kbyte/s. Sedangkan rerata kecepatan kompresi untuk *Static Arithmetic coding* hanya 78458 kbyte/s dan rerata laju kompresi *Adaptive Arithmetic Coding* tertinggi 24829 kbyte/s. Hal ini bisa dimengerti karena algoritma ini relatif lebih sederhana dalam proses komputasi. Namun konsekuensinya rata-rata rasio kompresi yang dihasilkan adalah paling rendah. Algoritma ini paling tepat jika digunakan untuk mengawali proses kompresi pada teknik kompresi bertingkat.

KESIMPULAN

Beberapa kesimpulan yang dapat diperoleh setelah menyelesaikan proyek ini adalah sebagai berikut.

1. Algoritma *Adaptive Arithmetic Coding* memberikan rerata rasio kompresi yang lebih baik sebesar 73% pada semua *file* sampel dibandingkan algoritma kompresi yang telah ada, yaitu RLE, Hufman, dan LZW.
2. Algoritma Adaptive Arithmetic Coding akan menghasilkan *file* kompresi dengan rasio kompresi maksimum pada pengkompresian *file* *.bmp, *.cpp, *.txt, dan *.xml sebesar lebih dari 81%.
3. Algoritma RLE rerata rasio kompresinya 19% dengan rasio kompresi tertinggi sebesar 56% pada kompresi *file* *.bmp. Huffman memberikan rerata rasio kompresi 49% dengan rasio tertinggi 70% pada kompresi *file* *.bmp. LZW menghasilkan rerata rasio kompresi 69% dengan rasio kompresi tertinggi pada kompresi *file* *.bmp sebesar 84%.
4. Kemampuan kompresi dengan rasio kompresi maksimum akan memberikan dampak yang berkebalikan terhadap laju kompresinya. Rerata laju kompresi untuk algoritma Static Arithmatic Coding 78458 Kbyte/s. Hal ini akan menurun lagi pada algoritma Adaptive Arithmetic Coding sebesar 24829 Kbyte/s. Sedangkan laju kompresi pada RLE mencapai 984164 Kbyte/s.

DAFTAR PUSTAKA

Gilbert, Held, 1991, *Data Compression – 3rd edition*, John Willey & Sons, Ltd.

Herianto, Tjendri, 1995, *Tuntunan Praktis Pemrograman C++*, Elex Media Computindo

Lempel, A., 2005, A Universal Algorithm for Sequential Data Copmression, *IEEE Transaction on Information Theory*, IT-23(3):337-343.

Lelewer, Debra A., Daniel S. Hirschberg,4/1/2002, *Data Compression*, Situs internet, URL: http://www1.ics.uci.edu/~dan/pubs/DC-sec3.html.

Nelson, M., Jean-Loup Gailly, 1996, *The Data Compression Book, 2nd edition*, M&T Books, New York.

Phamdo, Nam, 2002, *Data-Compression.com (internet site)*, Department of Electrical and Computer Engineering State University of New York Stony Brook, NY 11794-2350. URL: http://www.data-compression.com.

Salomon, D., Nam Phamdo, 2002, *Data Compression.com (internet site)*, Department of Electrical EngineeringState University of New York Stony Brook, NY 11794-2350.

Sedgwick, Robert, 1990, *Algoritma in C,* Addison-Wesley Publishing Company.

Wibowo, Wahyu C., 1991, *Pemrograman Berorientasi Objek*, PT. Elexmedia Komputindo, Gramedia Jakarta.

Wikipedia, 2006, Lossless Data Compression, wikipedia foundation, US.

URL: *http://www.data-compression.com*

URL: *http://www.howstuffworks/datacompression.html*

URL*: http://www.cblomm.com/src/lzw trie.zip*

URL: *http://datacompression.info/huffiman.shtml*

URL: *http://www.cs.mu.oz.au/alistair/abstractc/mnw98:acmtois.html*